A Research-Oriented Architecture for Providing Adaptive Collaborative Learning Support

Erin Walker[1], Nikol Rummel[2], Kenneth R. Koedinger[1]
[1]Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, USA
[2]Department of Psychology, Albert-Ludwigs University of Freiburg, Freiburg, Germany

Abstract. *Not written yet.*

1 Introduction

Over the past 30 years there has been an evolution in research on how students learn by collaborating, depicted in Figure 1 (Dillenbourg, Baker, Blaye, & O'Malley, 1995). Early work compared the effects of collaborative and individual learning, or looked at how the conditions of collaboration related to learning and attitudinal outcomes (see Slavin, 1996, for a review). However, to better understand the effects of collaboration, it is important to model collaborative interactions, which might include both collaborative task actions and verbal exchanges. Dillenbourg and colleagues (1995) termed this line of research the "interactions paradigm", where in addition to conditions of collaboration, interactions are characterized and related to outcomes. As it grew apparent that students often do not exhibit beneficial collaborative behaviors spontaneously, it further became relevant to determine how to support collaboration in order to produce the desired interactions, which would then hopefully lead to the desired learning outcomes (Strijbos, Martens, & Jochems, 2004). Much current collaborative learning research focuses on the effects of giving students *fixed* assistance, including declarative instruction on how to collaborate (e.g., Saab, van Joolingen, & van Hout-Wolters, 2007), examples of good collaboration (e.g., Rummel & Spada, 2005), and collaboration scripts that provide students with designated roles and activities as they work together (e.g., Fischer, Kollar, Mandl, & Haake, 2007). In evaluating these interventions, analyzing the collaborative interactions is critical both in determining if the interventions are having the desired effect on student behavior and in understanding how learning outcomes may result from the interactions. More recently, there has been a movement toward developing *adaptive* assistance for collaboration, where collaborative interactions are modeled as they occur, and the results of the analysis determines the content of the assistance given. Fixed scripts may provide students with too much structure and extraneous collaborative load, particularly for students who are capable of regulating their own learning (Dillenbourg, 2002). However, pre-collaboration training or teacher instruction may provide too little support for students during the actual collaboration, where students may not manage to

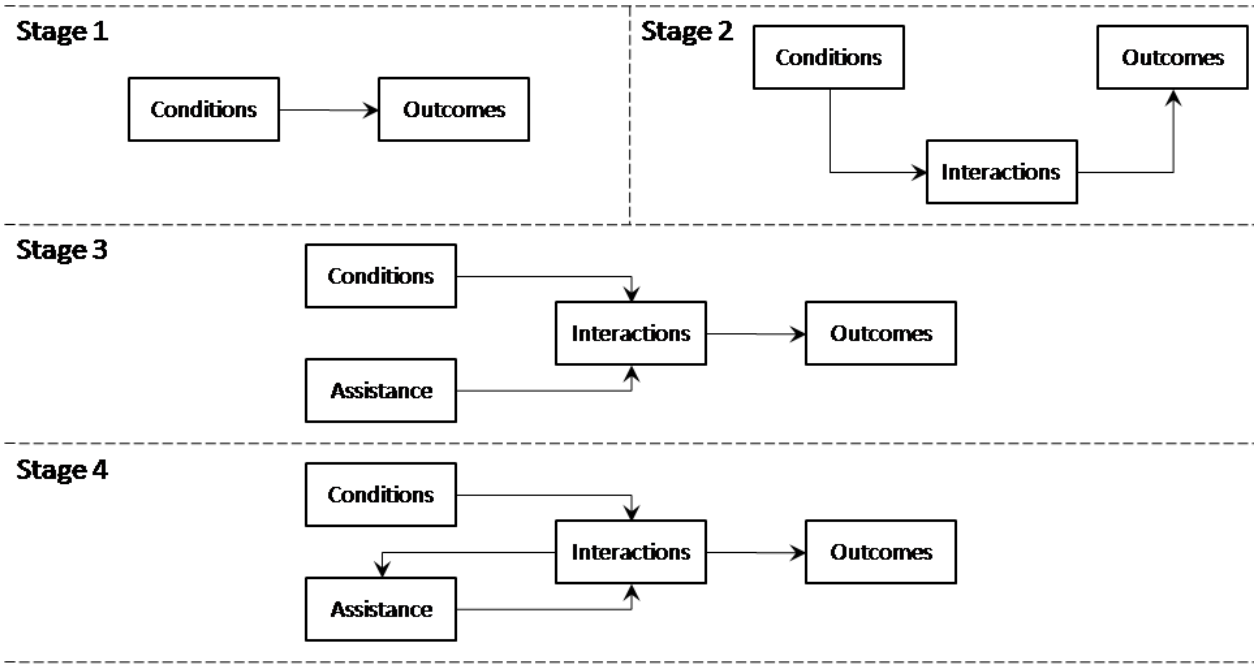

Figure 1. Four stages of research on collaborative learning.

follow the collaborative activity as designed (e.g., Ritter, Blessing, & Hadley, 2002). Therefore, adaptive support might be a better way of targeting the individual needs of students (Soller, Martinez, Jermann, & Muhlenbröck, 2005; Kumar, Rosé, Wang, Joshi, & Robinson, 2007; Rummel & Weinberger, 2008).

Although there are many potential learning sciences research questions surrounding the adaptive support of collaboration, this support has proven to be challenging to implement, and evaluations of adaptive support compared to fixed support have been promising but rare. The problem of delivering adaptive assistance to collaboration can be considered an instantiation of a more general assistance dilemma (Koedinger & Aleven, 2007), where in order to discover how to best deliver assistance to optimize student learning, one must manipulate the amount, type, and timing of help provided to students. In the case of collaborative learning, there are several levels on which assistance can be delivered, ranging from assistance on domain skills to assistance on elaborated verbal interactions. In cases where assistance on multiple levels might be appropriate at a single time, how best to integrate the different levels is an open question. It is resource-intensive to develop an adaptive intervention that implements feedback on a single level, never mind one that integrates feedback on multiple levels. Non-technological implementations of adaptive support for collaboration require an experimenter or a teacher to interact with each collaborating group (e.g., Braun, 2008; Hmelo-Silver, 2004; Gweon, Rosé, Carey, & Zaiss, 2006; Tsovaltzi, et al., 2008). This wizard of oz methodology is impractical for large scale research, let alone classroom deployment, and creates uncertainty as to whether different facilitators have different effects.

Instead, it may be advantageous to use computer-supported collaborative learning settings to examine the effects of adaptive support at different and possibly interacting levels. In these settings, task and language interactions can be automatically collected, guided, and used as input to a system that delivers adaptive feedback. Unfortunately, such systems take a long time to develop because of the difficulty of constructing accurate collaborative models and the challenges of having the system provide non-disruptive feedback to collaborating students. Although systems have been developed that provide adaptive support to collaboration, they rarely integrate assistance models and feedback on different levels (see Background section for a more detailed discussion). Further, the effects of adaptive feedback provided by these systems on collaborative interactions and learning outcomes has rarely been evaluated in large-scale controlled studies, despite the fact that the evaluations that have occurred have had promising results. For example, Kumar et al. (2007) found that adaptive support to collaborating pairs was better than no support to collaborating pairs and adaptive support to individual learning. It may be that removing some of the technical obstacles to implementing adaptive assistance conditions and relevant comparison conditions would encourage the further use of such systems to address learning sciences research questions.

In this paper, we introduce the Collaborative Tutoring Research Lab (CTRL), a research-oriented architecture for adaptive collaborative learning support that facilitates the collection of multiple streams of process data, the development and integration of assistance based on the data, and the implementation of relevant comparison conditions for experimental control. In the construction of the research architecture, we have adopted ideas from an individual learning perspective on delivering adaptive instruction: cognitive tutors, a type of intelligent tutoring system. Cognitive tutors are computer-based instructional systems that compare student actions to a model of correct and incorrect problem-solving and provide targeted feedback to students when needed. They have been successful at increasing learning in individual settings (Koedinger,

Anderson, Hadley, & Mark, 1997) and have evolved from acting as isolated interventions to serving as research platforms. For example, Project LISTEN's Reading Tutor supports the incremental addition and evaluation of features, and the collection of rich log data that can later be mined to provide insight into student learning processes (Beck, Mostow, & Bey, 2004). Cognitive tutors are often implemented using a component-based approach that facilitates the rapid development of research interventions (e.g., Koedinger, Suthers, & Forbus, 1999). Our CTRL architecture extends the individual tutoring scenario (one tutor, one student) to a collaborative multi-tutor setting (multiple students and multiple tutors, with different roles or for different purposes). Moreover, one of the strengths of our architecture is that it focuses on reusability: it facilitates the addition, removal, and integration of components. Adaptive collaborative conditions can be developed more rapidly by using existing computational models, and that comparison conditions can be created more rapidly by removing particular components of the adaptive system. We have used the CTRL architecture to create an adaptive support condition for a peer tutoring activity that integrates domain and collaboration assistance, and evaluated the adaptive support condition in a controlled classroom study by comparing it to a fixed support condition and an individual learning condition. Both control conditions were also implemented using the architecture. The results of the study increase understanding of the effects of adaptive support on peer tutoring, in part due to the rich process data collected, the nature of the adaptivity implemented, and the controlled comparison between conditions.

2 Background

In this section, we survey related work on adaptive collaborative learning support (ACLS). The types of systems of primary interest to us are *coaching systems*, as defined by Soller, Martinez, Jermann, and Mühlenbrook (2005) in their review of collaboration support systems. Coaching systems help students engaged in computer-mediated collaboration by assessing the current state of student interaction, comparing the current state to a desired state, and then offering assistance to the students. Coaching systems have a lot in common with intelligent tutoring systems, which also support students using the three phases of assessment, comparison, and assistance, but focus on individual learning. Moreover, intelligent tutoring systems, and cognitive tutors in particular, have moved away from merely being interventions and toward serving as research platforms to answer learning sciences questions about the effects of adaptive assistance. Our goal is to develop a similar research platform for ACLS. To this end, we focus our review in this section on ACLS systems that have been implemented and evaluated. The relevant systems are summarized in Table 1 and Table 2. In particular, the section looks at the kinds of interactions ACLS systems encourage, how adaptive feedback is delivered in these systems, how these systems are generally designed for reuse, and how the learning effects of these systems are evaluated. Further, we examine how cognitive tutor principles and architectures for individual learning might be able to contribute to our goal.

2.1 Interactions in Adaptive Collaborative Learning Support Systems

ACLS systems may support both collaborative task actions and computer-mediated conversation (see Table 1 for a summary of interactions enabled by ACLS systems). Often,

student interactions are structured either using micro-scripts, which operate on an action-by-action basis, or macro-scripts, which operate on the level of phases of activity (Dillenbourg & Hong, 2008). In our work, we are most interested in micro-scripts, or structuring interactions within a phase of collaborative activity. ACLS systems tend to include a shared workspace where students can work together toward a domain goal. Micro-scripts are often applied to these shared workspaces by giving students different roles in the workspace or by allowing them only to act at particular times. For example, as summarized in Table 1, COLER contains a shared workspace where students can collaboratively construct entity-relationship diagrams by interacting with coupled nodes and edges (Constantino-Gonzales, Suthers, Santos, 2003). Students have to indicate their intention to draw in the workspace, and when one student is drawing the other students cannot. Often learning systems that have a shared workspace also include a private workspace that contains no coupled objects, so that students can do individual work. The other primary component of many implemented ACLS systems is a text-based tool that allows students to communicate with each other in natural language. Within these tools, micro-scripts are often applied through the use of sentence-starters that students select to begin their utterance (e.g., "I would like to explain that…") or classifiers that student select after typing their utterance (e.g., "Give an Explanation"). As described in Table 1, Group Leader currently has 46 sentence classifiers that represent 10 subskills students should be exhibiting while collaborating, such as "Task Leadership" (Israel and Aiken, 2007). Finally, interfaces may contain widgets such as buttons through which the students can get information from the intelligent system. For instance, students can request four different types of help from HabiPro: clues to the solution, a worked example of the current problem, a worked example to a different problem, and the solution to the problem (Vizcaino, Contreras, Favela, & Prieto, 2000). Assuming that most current ACLS systems are logging all the actions that they enable, the systems capture collaborative task actions, verbal interactions, and meta-interactions that arise as a result of following micro- and macro- interaction scripts.

The interactions in an ACLS system can be viewed through the lens of "making thinking visible", which is a principle employed in cognitive tutor development (Koedinger & Corbett, 2006). In cognitive tutors, students are asked to perform several steps to complete each problem-solving task. These steps can be considered as subgoals in the problem-solving process. When the steps are explicitly represented in the interface, the subgoals become more salient to students, increasing their learning. In turn, when students take action in order to meet the subgoals, an adaptive system gains more insight into student cognitive processes than it would if students were simply providing the answer to the problem. For example, in the PACT geometry tutor students are asked to solve geometry problems and explain their steps using a menu-based interface (Aleven & Koedinger, 2002). The act of self-explanation is both beneficial for students and helpful for the cognitive tutor in identifying the source of student error. Scripts imposed on collaborative learning activities can have a similar function. Adding sentence starters to an interface can make a student's communication intention visible in addition to informing students about the communication acts expected (e.g., Israel & Aiken, 2007). Having private and shared workspaces in a system can make the discrepancy between an individual's private reasoning and their contributions to the group visible (e.g., Constantino-Gonzales, Suthers, & Santos, 2003) and thus provide input to an adaptive system.

Table 1. Task and assistance elements of ACLS

System	Task Elements	Assessment Method	Tutoring Goals
CARDDALIS			
COLER	Modeling, shared & private workspace, chat (classifiers)	Solution structure, individual contributions	IP, RI
COLLECT-UML	Modeling, shared & private workspace (phases), chat (classifiers)	Solution structure, individual contributions, solution quality	IP, RI, DM, DC
COMET	Medical problem-based learning in shared workspace, chat (unstructured)	Action counts, action sequences, student expertise, solution quality	IP, RI
CycleTalk	Shared workspace (different phases), unstructured chat	Chat counts, keywords in chat, parsing of chat	RI, TO, DC
Group Leader	Programming with chat (sentence openers)	Count dialogue acts, keywords, sequences of disagreement	RI, DM, RC
HabiPro	Editing computer programs using chat, shared workspace	Solution quality, individual expertise, help type requested, chat counts, keywords	IP, TO, RI, DC
LeCS	Case study (phases) in chat (sentence openers), shared text editor, solution representation	Length of time to complete a step, chat counts, solution	RI, DC
MArCo	Graphical planning in shared workspace, chat (dialogue games)	Logical conflict between student utterances	RC
OXEnTCHÊ	chat (sentence starters)	Chat counts, keywords	TO, RI

2.2 Modeling and Feedback in Adaptive Collaborative Learning Support Systems

Current ACLS systems assess collaboration based on targeted aspects of student interactions, compare the assessment to ideal collaborative qualities, and then provide feedback based on the comparison (see last three rows in Table 1 for an overview). In many ways, these ACLS systems are very different. Feedback policies with respect to both collaboration and domain feedback varies; some feedback is triggered by user actions (Tedesco, 2003), some is triggered by user inaction (Constantino-Gonzales et al., 2003), some is provided on demand (Vizcaino et al., 2000), and some is only provided when a user submits a solution (Baghaei et al., 2007). The representation of ideal student performance also varies between systems, ranging from finite state machines (Israel & Aiken, 2007) to decision trees (Constantino-Gonzales et al., 2003) to constraints (Baghaei, Mitrovic, & Irwin, 2007). Despite these differences, ACLS systems have broad commonalities with respect to collaborative skills targeted and how the skills are assessed in the context of the system. In fact, the types of support provided by ACLS can be interpreted using a collaboration analysis scheme developed by Meier and colleagues (Meier, Spada, & Rummel, 2007), where student interaction is rated on 9 dimensions. Some systems attempt to improve student interaction on Meier et al.'s dimension of *information pooling* (IP), i.e. how much students share their knowledge with their groupmates (Constantino-Gonzales et al., 2003; Baghaei et al., 2007). As represented in Table 1, assessment on this dimension is drawn from workspace actions: Student actions in a public workspace are compared to their actions in a private workspace in order to evaluate how much of their individual actions they are sharing with the group. Some systems instead focus on Meier et al.'s dimension of *dialogue*

management (DM), or how students execute conversational acts. Assessment in this area obviously focuses on chat actions, where sentence classifiers can be used to count utterances of particular types or even create a model of the student dialogue acts and compare it to a sequence of ideal dialogue acts. Then, drawing from earlier analysis systems such as EPSILON (Soller, 2004), the ACLS system can give feedback to students based on their contributions (e.g., Israel & Aiken, 2007). Some of the systems described in Table 1 help students in *reaching consensus* (RC; encouraging students to engage in productive conflict) by detecting and responding to loops of disagreement. There is also a growing trend toward using machine learning to classify student utterances instead of (or in addition to) sentence starters, with some success (e.g., Kumar et al., 2007). These efforts have mostly focuses on *task orientation* (TO), making sure students stay on topic. Up until now, we have discussed supporting either workspace actions or chat actions. Even systems that use metrics of assessment that might apply to both types of interactions often focus their analysis on either one. For example, a common dimension targeted for assistance is *reciprocal interaction* (RI), or whether everyone in a collaborative group is participating. Systems track actions in the shared workspace (e.g., Constantino-Gonzales et al., 2003), chat contributions (e.g., Viera et al., 2004), or the length of time since students have contributed last (Rosatelli & Self, 2004) in order to assess this dimension. However, systems do not generally use all three metrics.

The current functionality of ACLS systems, as described in the above paragraph, points toward potential future opportunities for advancement. Task-related feedback is not often integrated with collaboration feedback, even when it would make sense to do so. COLLECT-UML (Baghaei et al., 2007) provides students with both task-related feedback on the quality of their group solution and prompts to contribute elements from their individual solutions to their group solution. However, the system does not show sensitivity to whether the elements students have not shared with the group are correct or incorrect. This knowledge would augment the system's capabilities to provide relevant feedback: the system could suggest that students only share the correct elements with their group, or even suggest that students ask their group why an element in their individual solution is incorrect. Even assuming that the focus of some ACLS systems is on providing collaborative feedback only, these systems could be improved by integrating different streams of data and the types of modeling they provide. COLER (Constantino-Gonzales et al., 2007), for example, counts workspace actions to assess individual contributions but ignores chat actions. This separation might make it difficult to get the full picture of when feedback should be provided. Finally, the different types of feedback (collaborative and/or task-related) provided by ACLS systems are often kept separate by design, with prompts for each type of feedback appearing at different times during the student collaboration. For instance, GroupLeader (Israel & Aiken, 2007) has three types of feedback: get back on topic, incorporate a single idea per post, or re-evaluate a conflict. There is never a case where two types of feedback are given at the same time. Although this configuration is a good initial policy, as models become more complicated and better integrated, keeping the feedback separate in this way will not scale. One next step in ACLS design is to provide more complex assistance by integrating different types of models of interaction (both collaborative and domain), and different types of feedback.

Research on cognitive tutors (and intelligent tutoring systems more generally) has recently begun to explore the integration of different forms of assistance, in particular augmenting task-related feedback with metacognitive feedback. There has been growing recognition that the limitations of intelligent tutoring systems might be addressed by providing

students with metacognitive instruction, with the goal to enable them to regulate their own learning (Azevedo, 2005). One example of an existing metacognitive tutor is iSTART, a tutor for helping students to acquire reading comprehension strategies (McNamera et al., in press). iSTART asks students to explain a text to themselves as they read it, and then provides them with feedback on the type and quality of their explanations. In some tutoring systems, not only is metacognitive support provided, but cognitive and metacognitive tutoring are integrated. Output from a domain-specific cognitive model serves as input to a domain-general metacognitive model, resulting in more effective metacognitive model and better integrated feedback. One example of a tutor which uses this technique is the Help Tutor, which is a meta-cognitive tutor for help-seeking that is designed as a domain-independent addition to any cognitive tutor (Aleven, Roll, McLaren, Ryu, & Koedinger, 2005). The Help Tutor uses both student actions and information from the cognitive tutor to evaluate student help-seeking while problem-solving. For example, a student that attempts a problem-solving step (student action) with too low of a skill assessment for that step (cognitive tutor assessment) has committed a help-seeking error. Only one type of feedback is given at a time; if both the Help Tutor and the regular cognitive tutor have feedback to give to the student, the feedback source is chosen based on the type of student action and correctness of student action. Other researchers have explored similar methods of augmenting an intelligent tutoring system with agents that improve student motivation (Del Soldato & du Boulay, 1995), discourage students from gaming the system (Baker et al., 2006), or facilitate learning by teaching (Biswas et al., 2005). As collaboration can be thought of as a collection of metacognitive skills, the techniques for integrating metacognitive and cognitive tutoring could potentially be leveraged to combine collaborative with cognitive tutoring.

2.3 Implementation of Adaptive Collaborative Learning Systems

Many coaching systems (Soller et al., 2005) use a component-based architecture, which can enable the easy modification of an existing system and the reuse of system modules in novel configurations. In component-based architectures, software is divided into abstract components that can be specified to suit the developer's needs and flexibly integrated with other components using a standard framework (Krueger, 1992). At a minimum, the way a system is divided into components has a positive impact on reuse, because a single component can be enhanced or replaced without having to modify the other components. As ACLS systems are distributed applications with multiple users, one common implementation of these systems follows a client-server architecture, with an interface client provided for each student and a central server containing multiple components responsible for managing the collaborative sessions (e.g., Baghaei et al., 2007; Tedesco, 2003; Vizcaino et al., 2000). Collaboration between interface clients is often facilitated using a "what you see is what I see" policy, where objects are coupled in shared workspaces so that an action taken on a coupled object in one user's client is broadcasted to the parallel objects in collaborators' interfaces (Suthers, 2001). Similarly, text-based interaction tends to follow a traditional instant messaging format, where everything a user submits as an utterance is seen by their partners (e.g., Viera et al., 2004). The tutoring functionality of these systems is then generally located on the server. Many systems subdivide the tutoring module into different components, and although the components are named differently across systems, the underlying purpose is parallel across systems. ACLS systems generally include an *expert model*, which compares student actions to an ideal model of

collaboration, and a *feedback model*, which contains the logic for how feedback should be delivered to students (e.g., Kumar et al., 2007; Israel & Aiken, 2007; Tedesco, 2003). The two components handle all types of support the system offers. For instance, in the case of COMET, they would support both information pooling and reciprocal interaction (Suebnukarn & Haddaway, 2004). One or more *translator* components are also sometimes included to convert the low-level user actions into high-level representations of their collaboration that can be input to the expert model (e.g., Kumar et al., 2007, Israel & Aiken, 2007, Viera et al., 2004). A variation of this approach to developing a tutoring module is to include both individual expert models and a group expert model on the server, with the group model being either a parameterization of the individual models (Hoppe, 1995) or containing its own specifications for good collaboration (Baghaei et al., 2007).

Based on this description of components, the reuse facilitated primarily involves the ability to modify one aspect of tutor functionality without altering other aspects of tutoring functionality. For example, Kumar and colleagues (2007) how their expert model, translator, and feedback model are separate from each other, such that each component then can be iteratively improved without altering any others. Another way to facilitate reuse is by adding new components directly to existing configurations: In COLLECT-UML (Baghaei et al., 2007), group modeling components are added to augment the individual modeling components already present. Once an integration framework has been developed for the components, they can be more easily substituted for one another or combined in novel ways. For instance, Mühlenbrock and colleagues have created an integration framework where individual user interfaces register with the DALIS server, which then invokes a pre-specified set of support agents (Mühlenbrock, Tewissen, & Hoppe, 1998). Essentially, the DALIS server acts as the facilitator in a federated system (Genesereth, 1997). Similarly, LeCS treats tutors as clients, with a central facilitator managing the interaction between tutor clients and interface clients, although with no explicit integration framework (Rosatelli & Self, 1994). Although the described designs for reuse can make it easier to increase the complexity and sophistication of a single type of adaptive support, they do not necessarily facilitate the integration of multiple types of adaptive support and the efficient implementation of comparison conditions. Few ACLS systems specifically include multiple tutor components which each provide a different level of tutoring. One exception is COLER, which includes three expert model tutoring components: a “Participation Monitor”, a “Difference Recognizer”, and a “Diagram Analyzer” (Constantino-Gonzales et al., 2004). This division of tutoring components by functionality can make it easier to incrementally add tutoring complexity, particularly if there is a framework in place so that new tutoring models can be integrated with existing tutoring modelsh. If different tutoring subcomponents are supported, as in the architectures described above, the complexity of a single tutoring level can be increased. However, if multiple types of tutors are supported, then different models and feedback can be more easily integrated.

Cognitive tutor architectures are structured so that custom-built interface and tutor components can be integrated with existing components. This type of reusability can be found in Ritter & Koedinger’s (1996) component-based framework for facilitating the development of intelligent tutoring systems. Framework components are divided into tools and tutors, and a standard protocol for interchanging messages is defined to make it easier to swap different components in and out. So that off-the-shelf components can be used, the framework also includes a translator component to convert messages sent from the off-the-shelf components into the standard format, and convert messages sent to the component into a format that it

understands. Although Ritter & Koedinger demonstrated how the framework could be used with two separate tutoring applications, their emphasis was on the use of off-the-shelf applications for individual tutoring, rather than on the addition of metacognitive or collaborative components. However, further iterations of the cognitive tutor (e.g., the Help Tutor) have experimented with using a similar framework to add metacognitive tutoring; the Help Tutor module was added to the traditional cognitive tutor, and feedback from the two tutor modules were integrated as necessary (Aleven, McLaren, Roll, & Koedinger, 2006). Allowing multiple tutors, and providing an integration framework for the tutors, would allow us to provide more complex tutoring to students.

2.4 Evaluation of Adaptive Collaborative Learning Systems

Much of the evaluation of ACLS systems has been conducted on the technological aspects rather than on the effects of the assistance on student interactions and learning outcomes. See Table 2 for a summary of the evaluations that have been conducted on ACLS systems. In some cases, a technological evaluation meant evaluating the effectiveness of the collaborative assessment. For example, Mühlenbrock et al. (2004) in evaluating CARDDALIS described how well the model represented the student interactions. In other cases, it meant evaluating the predictive power of the models used. COMET used kappa to demonstrate the relationship between expert-constructed group solutions and system-predicted group solutions, with positive results (Suebnukarn & Haddaway, 2006). Finally, sometimes feedback itself was evaluated. For an evaluation of COLER, 73% of the advice the system provided to collaborative students was rated as “worth saying” by an expert. Research that has not focused directly on validating the system technology has tended to fall under the category of design experiments rather than controlled studies. To inform the development of the adaptive component of LeCS, data from dyads interacting using the LeCS interface were collected and analyzed (Rosatelli & Self, 2004), and after OXenTCHÊ had been implemented, the usability and the benefits of the assistance from a student perspective were rated (Viera et al., 2004). The few full studies that have been conducted using adaptive systems have been promising. As described in Table 2, to evaluate COLLECT-UML, Baghaei and colleagues (2007) compared an adaptive collaboration support condition to a no collaboration support condition and found that while there were no differences in domain learning gains, the experimental condition gained more collaborative knowledge. Even more encouraging was the study conducted by Kumar and colleagues (2007), which manipulated two variables: adaptive versus fixed support, and collaborative versus individual learning. They found that the adaptivity and collaboration interacted to produce a significant learning result compared to the other conditions. As the technical merits of the reviewed systems have been established, a logical next step will be to investigate the potential learning benefits of these systems, and there is reason to believe that these benefits exist.

In addition to taking principles from intelligent tutor design, building components on top of an existing tutoring system might accelerate the evaluation process. There are several obstacles to conducting controlled experiments with adaptive collaborative learning systems. Data is often required to fine-tune the system parameters, but it can be difficult to collect. After expending all the effort it takes to build an adaptive collaborative system, it can be too time-consuming to build appropriate control conditions for evaluation. Finally, once appropriately calibrated conditions exist, it can be difficult to find enough participants for the study, and even

more difficult to conduct the study in an ecologically valid setting. As intelligent tutoring systems are older than ACLS, there exists more infrastructure surrounding these systems that can facilitate evaluation studies. The Cognitive Tutor Algebra, for example, can be found in thousands of schools across the US, and therefore vast amounts of data are logged every day (www.carnegielearning.com). Tutor data is often mined in service of investigating learning science hypotheses and ultimately informing the improvement of intelligent tutoring systems (Beck, Mostow, & Bey, 2004). Similarly, it has become common practice to perform embedded experiments, making small modifications to already deployed tutoring systems (Mostow & Aist, 2001). Finally, because the tutors are so widespread, there are well-established relationships with schools that can be leveraged to gain access to classrooms and ecologically valid participants. For example, one of the goals of the Pittsburgh Science of Learning Center (PSLC) is to connect researchers and classrooms, and then instrument those classrooms so that it is easier to collect data and evaluate learning interventions. Developing ACLS systems on top of existing intelligent tutoring systems holds great promise both in making such systems more available and in using them as a platform for research on users' interactions, collaborative learning, and methods for adaptive support of collaboration and collaborative learning.

Table 2. Evaluation of ACLS support

System	Evaluation Purpose	Evaluation Specifics
CARDDALIS?		
COLER	Technological	Expert ratings of system support, comparison of expert & system support
COLLECT-UML	Controlled Experiment	2 conditions (adaptive collaboration support vs no collaboration support), classroom study, effects on learning and interactions
COMET	Technological	Predict individual & group solution paths
CycleTalk	Controlled Experiment	2 (collaborative vs individual) x3(adaptive vs static vs no support) design, classroom study, effects on learning & interactions
GroupLeader	Technological	Assess student dialogue acts
HabiPro	Technological	Assess student need for assistance, off-topic behaviors, & passivity
LeCS	Design Experiment	Students use a non-adaptive system to inform design
MArCO	Quasi-Experiment	Compare adaptive and non-adaptive pairs in lab
OXEnTCHÊ	Design Experiment	Usability, student ratings of system assistance

2.5 Summary

There is an opportunity in ACLS systems design to incorporate ideas from cognitive tutoring in order to shift the implementation focus from development platforms to research platforms. Our goal is to build a research platform for ACLS that facilitates the representation of rich interactions, the integration of different tutoring types, and the efficient creation of valid comparison conditions for controlled studies. Up to this point, ACLS systems have done a very good job at focusing their support at separate types of interaction, but have not generally integrated support based on different streams of input. The architectures that have been developed to make ACLS systems easier to implement have not emphasized the use of pre-existing tutoring modules as input to custom-built models, which would increase the potential ability of the tutoring system to provide assistance. These architectures have also not explicitly facilitated the creation of comparison conditions, which would increase the effectiveness of the empirical evaluation of the system in order to investigate learning sciences research questions. We see an opportunity here to develop an architecture for ACLS with the goal of facilitating controlled research into different types of adaptive support, and for this purpose we introduce the Collaborative Tutoring Research Lab (CTRL). This architecture focuses directly on the interaction between collaborating students and intelligent support, and would therefore ideally be used in combination with other approaches. For example, the tool-level integration provided by Freestyler (Hoppe & Gallner, 2002) or CoolModes (Pinkwart, 2003) would be a good complement for the tutor-level integration we facilitate. Additionally, CTRL would be a good fit as part of a higher-level integration platform such as SAIL (), which facilitates the authoring, deployment, and assessment of learning activities. The distinct contribution of CTRL is the establishment of an integration framework for pre-existing and custom-built components to provide adaptive tutoring to collaborating students.

3 Architecture

The Collaborative Tutoring Research Lab (CTRL) provides a flexible integration mechanism for independent components to form an adaptive collaborative learning support (ACLS) environment. Using the architecture, the feedback from different tutors can be combined, meaning that students can receive complex tutoring based on multiple streams of process data. New tutor components can capitalize on existing tutor models, increasing the meta-cognitive tutoring possible. For example, a meta-tutor for sharing information with your teammate would be able to use results provided by a domain tutor about whether the facts shared were correct. The architecture facilitates the addition and removal of components in order to create appropriate comparison conditions for adaptive support. In this section, we outline the basic components involved in the architecture, the way they interact with each other, and the way they can be integrated. To ground our description, we use as a hypothetical example an adaptive support condition that might be implemented in CTRL, using a collaboration script developed by Rummel, Diziol, Spada, & McLaren (2007). The actual design, implementation, and evaluation of a peer tutoring scenario using CTRL is described in Section 4. In the Rummel et al. script, students first use a spreadsheet and graphing interface to individually solve a problem that contains a single equation, and then join together to collaboratively solve a more complex

problem incorporating both equations. While collaborating, students receive two types of feedback: cognitive feedback on their problem-solving, and collaborative feedback encouraging them to elaborate on the help received from the intelligent system. After finishing a problem, students enter a reflective phase, where they evaluate their collaboration and set goals for improvement. This script was shown to improve deep conceptual learning, but was implemented face-to-face rather than using computer-mediated collaboration, limiting the breadth of the adaptive support that could be provided to the students. We illustrate our architecture using examples from a hypothetical computer-mediated implementation of the script. While we use this scenario as an example, it is important to emphasize that the architecture might apply to many different types of ACLS.

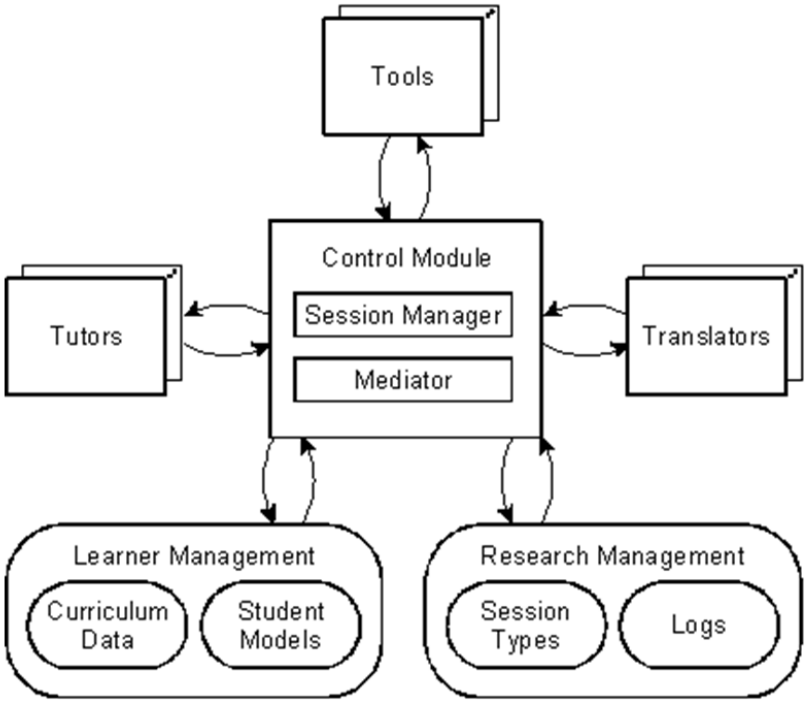

Figure 2. High level overview of CTRL. CTRL consists of tool, tutor, and translator agents, learner and research management data stores, and a central control module.

A high-level overview of our architecture is depicted in Figure 2. CTRL consists of six different types of components, based in part on Ritter and Koedinger's (1996) description of plug-in tool and tutor components:

1. *Tools*: Components used by the student to take problem-solving actions
2. *Tutors*: Intelligent components that provide students with problem-solving assistance
3. *Translators:* Facilitate inter-component communication and the implementation of collaboration scripts
4. *Learner Management:* stores curriculum information and student model data
5. *Research Management:* stores protocol logs and information about how the components involved can be integrated with each other
6. *Control Module:* constructs and manages collaborative sessions, both on a problem-to-problem level (session manager) and on a action-to-action level (mediator)

The focus of our architecture is on facilitating interactions between tool, tutor, and translator components, and we define and discuss each of those components in more detail in Section 3.1.

In Section 3.2, we describe how the various components communicate with each other. Section 3.3 outlines how the control module and research management store interact to allow the flexible integration of components and construction of multiple collaborative conditions. Learner management is not further described because it is outside the current scope of our architecture.

3.1 Component Functionality

A *tool* is a piece of software that a student interacts with in order to solve problems in a particular domain. A tool could be as simple as a text-editor that allows students to write essays or as complex as a simulation environment for chemical lab experiments. The CTRL framework allows for any number of tools to be involved in the learning scenario. Multiple users can collaborate remotely while each one uses different tool components of the architecture. There is not necessarily a one-to-one mapping between students and tools; a single student could have access to multiple tools (e.g., an instant messaging tool in addition to the text-editor), and two students could conceivably be using the same tool at the same computer. However, we assume for the purposes of this discussion that in a condition with multiple users, a tool represents a single user's interaction with the system as a whole. In the scenario used by Rummel and colleagues, there are two tool components, one for each collaborating student. Each tool component consists of a graphing widget, a spreadsheet widget, and an instant messaging client.

Tool components contain the interface to the user, a domain model, and meta-knowledge of tutoring. The interface is the point of interaction between the user and the tool (and by extension, the system as a whole). The domain model is present so that the tool can update its state without input from an additional component. For example, if the student types a formula into the spreadsheet, the domain model would compute the result of the formula and display it to the student. A user can then interact with a tool without input from any tutoring component, and therefore a tool is not bound to a given tutor. Although tools should be able to share domain models, this behavior is currently not explicitly supported by our architecture, in part because of our focus on using pre-existing components that probably already contain their own domain model (in line with the Koedinger, Suthers, & Forbus approach, 1999). Tools also contain meta-knowledge so that they can convert semantic feedback from a tutor agent into a format appropriate for display. For example, upon receiving the semantic message "cell A2 is wrong," the meta-knowledge model would convert it into "turn cell A2 red." Locating this knowledge in the tool means that tutors can send general messages rather than tool-specific messages, and can therefore be used with any tool. The functionality that we have described is ideal for integration into our architecture, but it is likely that many pre-developed tools we may want to use will not incorporate all functionality, and may be closed systems or difficult to modify. In these cases, we use *translator* components to compensate for the missing functionality.

Translator components are all-purpose facilitators that bridge communication between other components. They have two general functions. First, they make it possible to integrate components that do not conform exactly to the framework specification by providing missing functionality (e.g., an implementation of tutoring meta-knowledge) or by converting individual component messages into the standard message format. This aspect of translator functionality is very much in line with the translators discussed in Ritter and Koedinger (2006) and Kumar et al. (2007). Second, translators can impose a structure on the collaborative interaction, by communicating certain actions across tool components (such that a user action on one

component is displayed on all other relevant components) and by triggering changes related to collaboration scripts to the tool components. Like tools and tutors, there can be any number of translator components incorporated in a learning scenario. The specific implementation of a given translator would depend on its function. In the script by Rummel and colleagues, a translator could be used to allow graphing and worksheet actions made by one student to appear on the other student's screen. This approach, where translators facilitate collaboration, is different from the more traditional object coupling approach in CSCL systems (Suthers, 2001), where students can automatically see all actions made in a shared workspace. There may be cases during a student interaction where actions that would generally be collaborative should not be shared (e.g., when one collaborating student makes an error, it may not always be desirable to broadcast the error to group members). We chose this implementation so that a designer of a learning environment has more control over structuring the interaction between students.

Tutor components are any components that provide adaptive support to students, generally by comparing their actions to a model, providing assistance based on the model, and assessing their skills based on the model. For example, tutors might range from a domain tutor for writing grammatical sentences based on a constraint-based model to a metacognitive tutor for proofreading a paper based on a cognitive model. Any number of tutors can be involved in a learning scenario, and any type of tutor can be used in our framework. In the scenario used by Rummel and colleagues, there are two tutor components involved: a cognitive tutor that provides algebraic support, and a collaborative tutor that provides support for the student interaction. Tutor components should contain an expert model, a feedback model, and a student model. Like in regular intelligent tutoring system functionality (as described in Van Lehn, 2006), the expert model evaluates the student action, the feedback model determines the sort of feedback that is given, and the student model assesses the student performance (or in some cases, the group performance). As with tools, any pre-existing tutor components used that do not have the desired functionality can be augmented with a translator component.

3.2 Message Protocol

All components communicate with each other using a standardized set of messages, providing guidelines for the development of new components that can be incorporated into the framework (see Table 3). As components may be running on different machines, messages are sent remotely. In these messages, details specific to the implementation of individual components are hidden as much as possible and only abstract semantic content is communicated. In this paragraph, we will enumerate the high-level representations that form the parameters and return values of the messages sent, and in the following paragraph we will discuss the types of messages themselves. First, a *Student Interaction*, or a step that can be taken by a user in the interface, is represented using four parameters:

1. *Student* – the student taking the action.
2. *Selection* – the widget being acted upon.
3. *Action* – the action performed upon the widget.
4. *Input* – any additional information necessary for the action.

For example, in the graphing interface of Rummel and colleagues an action might be to change the location of a point on the graph by dragging and dropping it. Here, the selection would be the point being dragged, the action would be "changing location", and the input would be the new

Table 3. Message passing between components.

Message Name	Input	Output	Sending Components	Receiving Components
launchComponent	Component properties	Success or failure	Session Manager	Tool, Tutor, Translator
quitComponent	None	Success or failure	Session Manager	Tool, Tutor, Translator
getNextProblem	Problem-selection properties	Problem properties	Session Manager	Tool, Tutor, Translator
changeProblem	Problem properties	Success or failure	Session Manager	Tool, Tutor, Translator
processInteraction	Interaction	None	Tool, Translator	Tutor
scriptInteraction	Interaction	None	Translator, Tutor	Tool
processFeedback	Interaction, Response	None	Tutor, Translator	Tool
setProperty	Component property	None	Translator, Tutor	Tool, Translator, Tutor
getValue	Attribute	Value	Translator, Tutor	Tool, Translator, Tutor
putData	Data properties	None	Mediator	Learner Management, Research Management
getData	None	Data properties	Session Manager	Learner Management, Research Management

coordinates of the point. The concept of a selection-action-input triple can be traced back to Anderson and Pelletier (1991). A *Tutor Response* to a student interaction is represented by four parameters:

1. *Tutor* – the tutor sending the message
2. *Action Evaluation* – the type of message (e.g., correct, incorrect, highlight)
3. *Feedback Message* – any message the tutor wants to send
4. *Skill Assessment* – the change in student skill values

For example, a domain tutor might approve a student action (indicating it was correct), send a feedback message for encouragement, and increase the value of the relevant skill. For passing other data between components, we use a *Properties* parameter, which is a more conventional data structure containing any number of attribute-value pairs. As described in Table 3, information that is not a Student Interaction or Tutor Response (such as current problem details) is communicated as a set of Properties.

These data structures are then used as parameters and return values for the message types exchanged between components (see Table 3). For example, when a session is started a *getData* message would be used to retrieve relevant curriculum and student information, and *launchComponent* messages would be used to start and configure all the relevant components. While elements of this message protocol are taken from Ritter and Koedinger (1996), the protocol is more abstract than the protocol that they defined, in order to facilitate a variety of potential learning environment interactions. Because the problem-solving interactions are the core messages of our architecture, here we present a more in-depth example of how those messages might be used by the different components (see Figure 2). The example includes two tools (representing two collaborating students, Bo and Jan), two tutors (representing a domain tutor and a collaborative tutor), one translator to implement the shared collaborative workspace, one research management component, and the mediator subsection of the control model. In the example, the tool receives input from the user and sends information about the user action to the

control model, using a *processInteraction* message. Once the control module receives the message, it logs it, and then redirects it to all components that should receive it (in this case, the translator and the two tutors). The translator takes the message and transforms it into a *scriptInteraction* message in order to reproduce a student action on another interface, which is sent back to the mediator. Meanwhile, the domain (math) tutor evaluates the user action, and sends its feedback to the mediator, which passes it along to the collaboration (chat) tutor using a *processFeedback* message. The collaboration tutor, using the user action and the feedback as input, evaluates the action and sends its feedback back to the mediator using a *processFeedback* message. The mediator has now received messages from the translator, the collaboration tutor and domain tutor. The mediator integrates the messages, passes the scripting message along to both tools, and then sends the feedback message to Jan's tool. Although not all collaborative scenarios will operate in exactly this way, these messages form the building blocks for handling interactions between tool, tutor, and translator components.

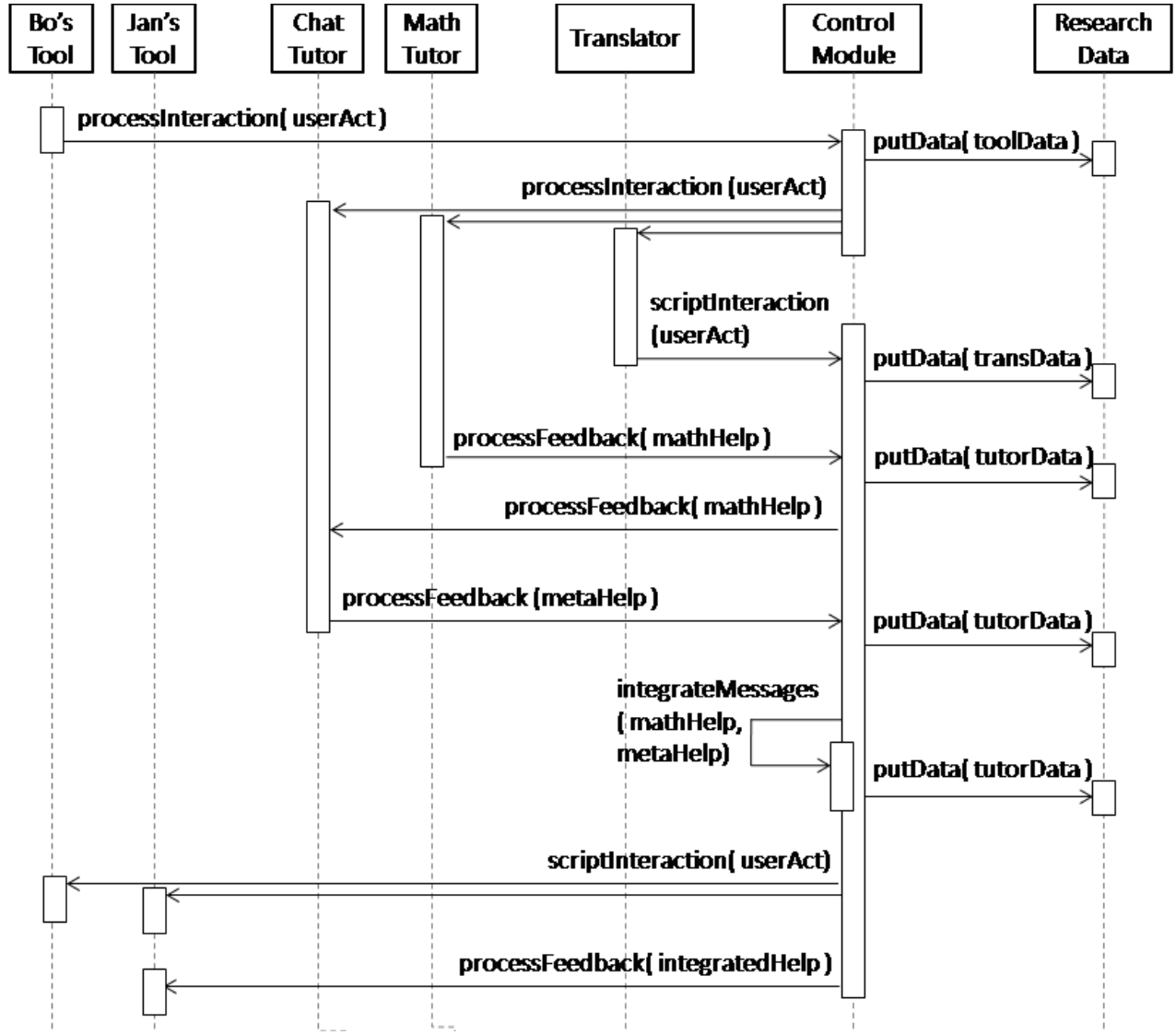

Figure 3. Message-passing between components.

We have explicitly chosen to leave some elements necessary for implementing a computer-supported collaborative learning system unspecified by the architecture, because they are outside of our main focus. The system may be used by students at different computer terminals, and as such, some components of the system (e.g. the control module) will be running on a central server, and some components (e.g. the tool components) will be running on various clients. However, the way components are distributed may depend on the deployment environment, so we leave it purposefully ambiguous. Also, because components are distributed, all messages need to be sent remotely, and we leave the implementation of the specific protocols up to the developer. Finally, to be deployed in a classroom, multiple sessions handling multiple student pairs need to be run at once, meaning that a server needs to handle client logins and launching the collaborative sessions. Although we do not outline general guidelines for accomplishing these goals, we do discuss our implementation of these features in Section 4.

3.3 Component Integration

In addition to illustrating how messages are passed between components, there are several notable elements of the above example which highlight the centrality of the control module during a session. All messages sent go through the mediator, which logs the messages prior to sending them to the relevant components. In this manner, the logging of different streams of interaction is combined within a single framework. Further, the mediator is in control of which components are involved, where messages get sent, and how messages are integrated. It is the central component that facilitates the flexibility of the architecture. Although the way we support collaboration is different from other systems, both collaboration and the integration of different tutors are facilitated by the control module. A translator component can be built to echo messages from one tool to another, facilitating collaboration. Additionally, the output of one tutor module can be used as input to a second tutor module and tutor messages can be combined in the mediator, facilitating the integration of different tutor components. While our architecture is not the only architecture to use a federated system (see Rosateli & Self, 2004; Muhlenbrock, Tewissen, & Hoppe, 1998), its unique contribution is that it focuses specifically on integrating different tutor components and on the efficient implementation of comparison conditions.

The central control module facilitates the integration of different components, helping to meet our goals of providing complex adaptive functionality and making it easier to create control conditions. In standard use of the intelligent tutoring system, each individual component has knowledge of where it is sending and receiving messages, and this configuration works because the system is so simple (the tutor sends messages to the tool, the tool sends messages to the tutor). With multiple components, a central body is needed to manage all the communication. The control module uses a representation of the session characteristics in order to determine how to route the messages. Each condition facilitated by the architecture is represented as a *session type* stored in research management. Each session type contains three arrays corresponding to three different types of components (tool, translator, tutor). Session types also contain a set of logical rules for how messages are passed between components. These rules can be as simple as:

IF a message *m* was sent by any tool
THEN send *m* to every tutor

However, some rules will need to be more complex, as they should also represent how to integrate feedback messages from different tutors. For example, if there is a participation tutor and a domain tutor involved in a session, a rule represented in the session type might be:

IF step *s* is incorrect
AND *m* is a domain feedback message
AND student *a* has not participated sufficiently
AND *n* is a participation feedback message
THEN aggregate *m* and *n* and send *m* + *n* to *a*.

Rules can involve any information available to the mediator, including the components involved in the message, the parameters of a particular message, curriculum or student parameters, and a pre-set priority of the message.

Once a session type has been created, the session manager and mediator can use it as a guideline for how different components should be interacting. When a collaborative session is started, the session is associated with a given session type. How this association is made is left open: it can be based on user login, or a particular curriculum, or even be selected by the user. The details of the particular session type discussed in the above paragraph are then retrieved from research management and stored locally in the control module. The session manager iterates through the components involved to send a high-level message (e.g., launching each component). The mediator's function is to control the low-level message passing between components by intercepting all messages sent by a component and directing them to the appropriate targets, following the rules outlined in the session type. Therefore, based on the session type activated, the same components can be used in different ways. Adding or removing a component can be as simple as creating a new session type, without the need to modify the other components involved in the interaction. Of course, depending on the complexity of the rules, authoring session types might be a challenge (particularly for non-programmers). In the discussion of an instantiation of our architecture in Section 4, we discuss the potential utility of rule templates for accelerating the authoring of session types.

The central control module also facilitates the creation of an integrated log of collaborative interactions. In the architecture, each semantically meaningful action occurring within a component is sent to the control module, which transforms the action into an xml string, and sends it to a data store in the learner-management component. In this manner, logs from each component are automatically integrated and can be reviewed together after a study without any further processing. The logging protocol of the architecture is based on the Pittsburgh Science of Learning Center Datashop logging protocol (learnlab.web.cmu.edu/dtd/), which records semantic-level messages sent from tool and tutor components. These tool and tutor logs follow the concept of a transaction described by Van Lehn et al. (2007), where a user action and the tutor response to the action are linked. In our framework, a processUserAction message is logged as "tool message" to the learner management module, with the Student Interaction parameters, a unique id, and a timestamp being represented in the log (see Figure 3 for a pictoral representation). A responding processFeedback message is logged as a "tutor message" to the learner management module, with the Student Interaction parameters, Tutor Response parameters, and a timestamp being captured. The relationship between the tool and tutor messages is also captured, as the tutor message contains the ID of the tool message that triggered it. Logs also include *context messages*, which are initiated by the control module, and record information about the problem being solved, the settings of the learning environment, or the

experimental design. Once a relevant context message has been logged, both tool and tutor messages will be linked to it, containing the context message id.

Because our architecture is designed for adaptive collaborative learning systems rather than individual intelligent tutoring systems, the logging supported needs to be broader than the protocol discussed by VanLehn et al. There needs to be an additional type of message supported: a *scripting* message, logged whenever a translator or tutor changes the problem state of a tool. In this case, the Student Interaction parameters, the timestamp, the id of the relevant context message, and the id of the relevant tool message is logged. Second, because the architecture supports multiple users on multiple tools, it is important not only to record the user of the message (part of the Student Interaction parameter), but the collaborative session that the user is involved in, and the role of the user within that session. In CTRL, we incorporate this information into the context message, which logs the learning environment settings. Third, because the architecture supports multiple tool responses, the relevant metaphor for analyzing the data is not a single tool-tutor transaction but a more complicated chorus of responses to a tool action. Not only does each tutor response need to be logged, but so does the final message constructed by the mediator to be sent to each tool.

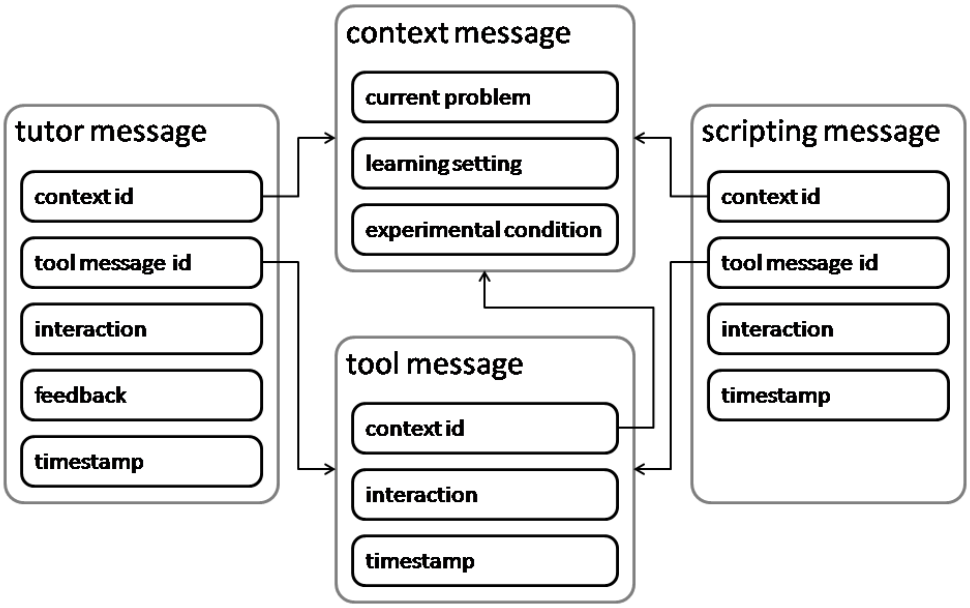

Figure 4. Logging format for student-tutor interactions.

3.5 Summary

Ideally, the CTRL architecture accomplishes the three goals of capturing rich process data, integrating feedback from multiple tutor components, and making it easier to implement comparison conditions. All semantic messages from components are sent to the control module, which creates a log of all student interactions including verbal interaction, collaborative problem-solving actions, and the intelligent tutor responses. Multiple intelligent tutors can be incorporated into the system (both pre-existing and custom-built) by changing the definition of a session type in the mediator. Domain-general intelligent tutors can use the output of domain-

specific tutors as input into their models. Integration of tutor-based feedback is facilitated using a series of rules specified on the session type. Finally, because it is relatively easy to modify session types and components are designed to be independent, it becomes possible to remove components from collaborative sessions in order to create multiple comparison conditions. In the following section, we discuss an instantiation of the architecture which demonstrates these positive features.

4 Evaluation

We evaluated the suitability of our CTRL architecture as a research platform by using it as the foundation for conducting an experiment on the effects of adaptive support in the context of a collaborative learning activity. In this section, we first describe how we designed an ACLS intervention in which we augmented a successful intelligent tutoring system, the Cognitive Tutor Algebra (CTA), with a peer tutoring activity. Our design drew on previous successful peer tutoring interventions and included both fixed and adaptive assistance. Second, we describe how we implemented the adaptive support condition, and two comparison conditions, using the CTRL architecture described in Section 3. We discuss places where the architecture was successful and places where lessons from our implementation informed the current architecture. Finally, we describe a controlled classroom study in which we compared the adaptive peer tutoring condition against the two comparison conditions. Our results benefitted from having access to process data, having an adaptive intervention that relies on both domain and collaboration models, and having strong comparison conditions.

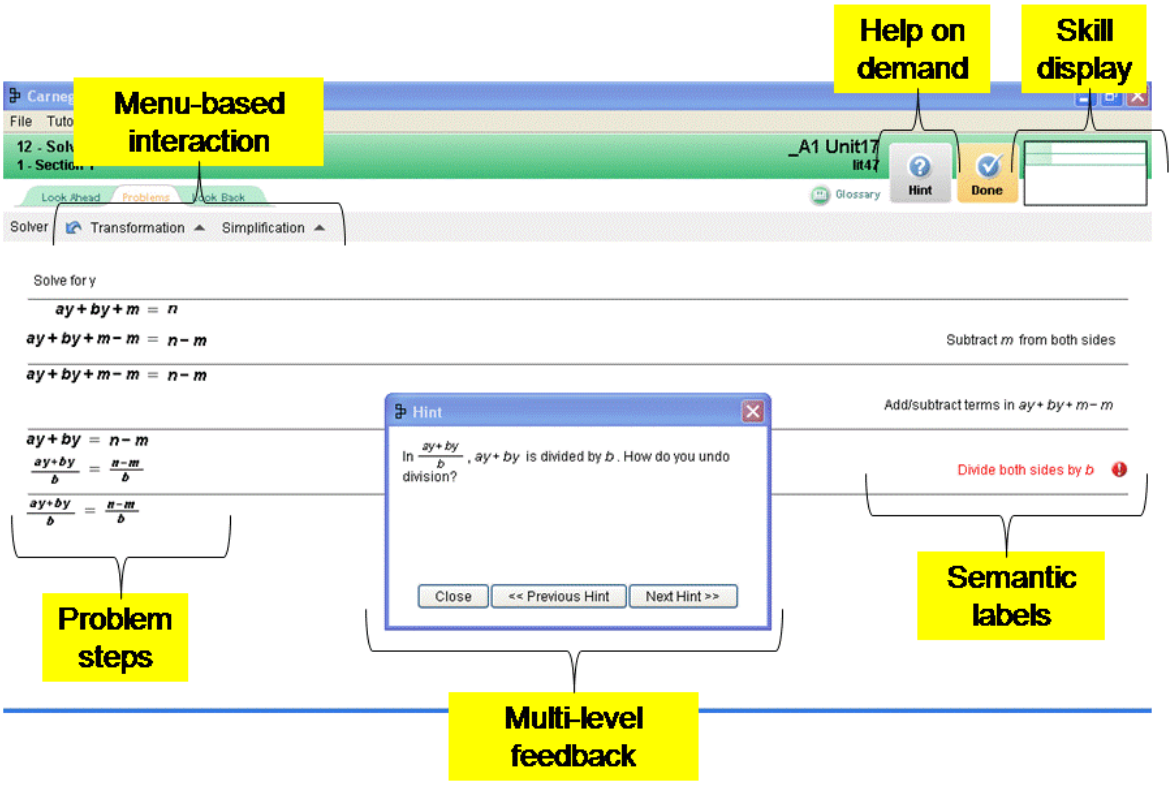

Figure 5. Individual version of the CTA

4.1 Intervention Design: Peer Tutoring in the Context of the Cognitive Tutor Algebra

Our intervention is designed as an addition to the Cognitive Tutor Algebra (CTA), which is an ideal candidate for being augmented by collaborative activities in many ways. Figure 4 shows the literal equation solving unit of the CTA. Students use the menus to manipulate the equation, selecting operations like “add x” or “combine like terms”. As they do so, the semantic label for the operation appears on the right hand side of the screen. For certain problems, students have to type in the result of the operation in addition to simply selecting the operation. As the students solve the problem, the CTA compares their actions to a model of correct and incorrect problem-solving behavior. If they make a mistake, they receive feedback in the interface; often they additionally receive a bug message describing their misconception. At any point, students can request a hint on the next step of the problem. The CTA monitors student skills and reflects them to the students in a skill display (Skillometer), in addition to selecting problems based on student skill mastery. As students may acquire shallow conceptual knowledge while using tutoring systems, recent research has augmented cognitive tutors with activities that encourage elaboration such as self-explanation or scripted collaboration. There are promising early results on the benefits of adding supported collaborative activities to the CTA compared to unsupported collaborative activities and individual learning (Rummel et al., 2007).

We augmented the CTA with a reciprocal tutoring script. When students act as peer tutors they benefit because they are engaging in knowledge-building, as they reflect on the current state of their knowledge and use it to construct new knowledge (Roscoe & Chi, 2007b). Because these positive effects are present even if peer tutors have low domain knowledge, researchers often implement reciprocal peer tutoring programs, where students of similar abilities take turns tutoring each other. This type of peer tutoring has been shown to increase academic achievement and positive attitudes in long-term interventions integrated into classroom practice (Fantuzzo, Riggio, Connely, & Dimeff, 1989). Biswas et al. (2004) concluded that three general properties are related to tutor learning: increased accountability for the domain material, reflection on the actions of the tutee, and elaborated interaction through asking questions and giving explanations. Tutee learning, on the other hand, appears to be maximized at times when the tutee reaches an impasse, is prompted by the tutor to find and explain the correct step, and is given an explanation if they fail to do so (Van Lehn, Siler, Murray, Yamauchi, & Baggett, 2003). Unfortunately, students do not often exhibit these beneficial knowledge-building behaviors spontaneously when tutoring (Roscoe & Chi, 2007a), and successful interventions have provided peer tutors with assistance in order to achieve better learning outcomes for both tutors and tutees. For one, this assistance can target *tutoring behaviors*. For example, having tutors ask tutees a series of questions at different levels of depth had a significantly positive effect on tutor learning (King, Staffieri, & Adelgais, 1996), as did training students to deliver conceptual mathematical explanations and give elaborated help (Fuchs et al., 1997). However, it is just as critical for assistance to target the *domain expertise* of the peer tutors, in order to ensure that students have sufficient knowledge about the correct solution to a problem to help their partner solve it. If peer tutors do not have this expertise, there may be both cognitive consequences (tutees cannot correctly solve problems; Walker, Rummel, McLaren, Koedinger, 2007) and affective consequences (when students feel that they are poor tutors they become discouraged; Medway & Baron, 1997. Domain assistance can take the form of preparation on the problems (Fantuzzo,

Riggio, Connely, & Dimeeff, 1989), access to the problem answers, and scaffolding during tutoring (Fantuzzo, King, & Heller, 1992).

In our peer tutoring design, we script the interaction to create conditions conducive to the display of positive *tutoring behaviors.* The script includes two phases: a preparation phase and a collaboration phase. In the *preparation phase*, students solve the problems that they will be tutoring, using the individual version of the Cognitive Tutor Algebra. After each problem, they answer a reflection question that prepares them to tutor on the problem, such as "What is a good explanation to give to your partner about a problem step?" Including a preparation phase helps to give students the domain knowledge necessary to later tutor their partner. Also, it may be beneficial for learning in itself, because the expectation of tutoring may lead students to feel more accountable for their knowledge and therefore attend more to the domain content during preparation. Partners are given different sets of problems to solve in the preparation phase. In the *collaboration phase*, students then take turns tutoring each other on the problems that they solved in the preparation phase. For example, if Bob and Sara are partners, and Sara was the tutor on the first problem, Bob would be the tutor on the second problem. Then Sara would solve the second problem just as though she was using the individual cognitive tutor, by manipulating the menus in the Equation Solver tool and typing in the results of a step when necessary. Bob in the role of the tutor cannot take actions in the problem himself, but he can see every step Sara takes on the problem and the results of every type-in entry. He can mark her answers right or wrong by clicking on them with a particular tool. He can also monitor her knowledge by raising or lowering the values of her Skillometer bars. These monitoring demands might lead Bob to reflect more on the knowledge required to solve the problem, and on his knowledge, by extension. Sara sees every action Bob takes to correct her or give her feedback on her knowledge. Bob and Sara can interact with each other in natural language using an IM tool. It was our hope that the chat would facilitate elaborated interaction between the students. Bob also has access to the problem solution in a tab in the interface, in order to provide him with assistance for the domain demands during tutoring (see Figure 5).

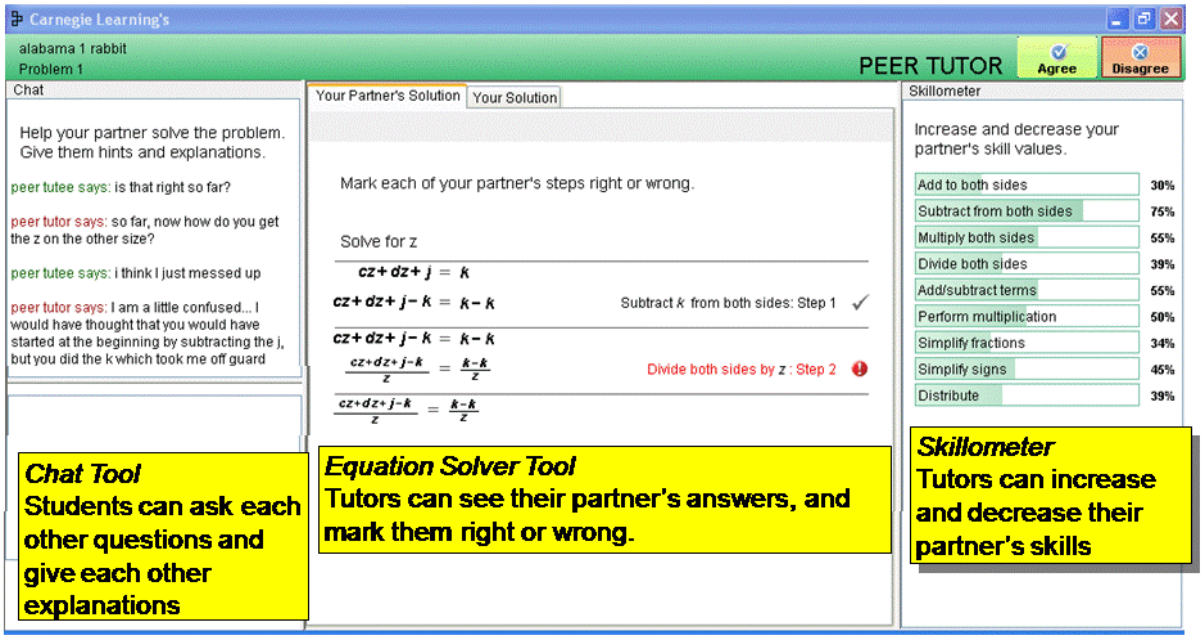

Figure 6. Peer tutor interface.

For our evaluation of the CTRL architecture, we used the CTA models to provide adaptive assistance to the peer tutor concerning *domain expertise,* using an intelligent system called a *meta-tutor*. In a pilot study with unsupported students using the peer tutoring script, we found that peer tutors had a lot of trouble giving correct domain help to their tutees, and tutees solved few problems correctly (Walker, Rummel, McLaren, Kodiner, 2007). Therefore, we focused our first attempt at adaptive assistance for peer tutor feedback to peer tutee problem-solving actions. There are three main ways a peer tutor can provide this type of feedback to the peer tutee:

Path 1. Responding "agree" or "disagree" whenever the tutee clicks the done button

Path 2. Marking a problem step "right" or "wrong" after the tutee has taken that step

Path 3. Providing a hint in the chat window

For Path 1, the ideal model of performance is that whenever the tutee indicates he is done with the problem, the peer tutor clicks "agree", and whenever the tutee is not actually done with the problem, the peer tutor clicks "disagree". Similarly for Path 2, the ideal model of performance is the that peer tutor marks a step right when it is in fact correct, and marks a step wrong when it is incorrect. Path 3 is a little more complicated, but for the purposes of this discussion the ideal model would simply be that the tutor provides a correct hint in the chat window. In the context of Path 1 and Path 2, the meta-tutor provides feedback whenever the peer tutor deviates from the model (e.g., whenever a step that is actually correct is marked wrong). All feedback is given to the peer tutor, with the hope that peer tutors will deeply process the feedback as they attempt to communicate it to the tutee. The goal of providing the feedback is not simply to force the peer tutor to reproduce every action the CTA would have provided. The meta-tutor provides feedback to peer tutors based on their actions, not their inaction; so if the peer tutee does something wrong and the peer tutor does not respond, no action on the part of the meta-tutor will be taken. In order to support Path 3, we make help-on-demand available to the peer tutor. The peer tutor can ask for a hint at any time, and use it as a basis for assisting the peer tutee. Both hints and feedback always include a prompt for students to collaborate, and the domain help peer tutees would have received had they been solving the problem individually (see Figure 6).

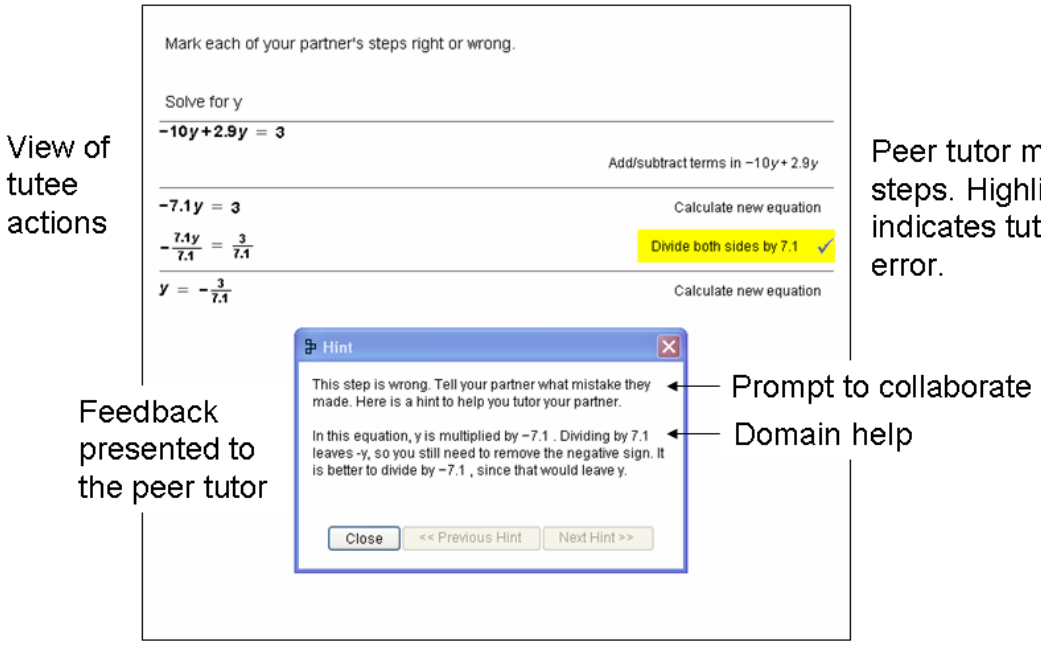

Figure 6. Peer tutor interface.

Our learning sciences research goal was to evaluate the effects of the adaptive assistance by comparing it to two comparison conditions (see Figure 7). We introduced a close comparison condition where students received fixed assistance, and therefore only whether students received adaptive support from an intelligent tutor compared to simply the problem answers was manipulated. We also used a far comparison condition representing current classroom practice, where students used the cognitive tutor individually and as they would during their regular curriculum. We hypothesized that the adaptive support condition would be more effective at increasing learning than the fixed support condition because the support is provided to peer tutors only when needed. Further, collaborative learning should be better than individual learning because students have the opportunity to interact about the domain material in depth. In the following section we describe how we implemented our three study conditions using the architecture described in Section 3. Then we present the empirical study and its results.

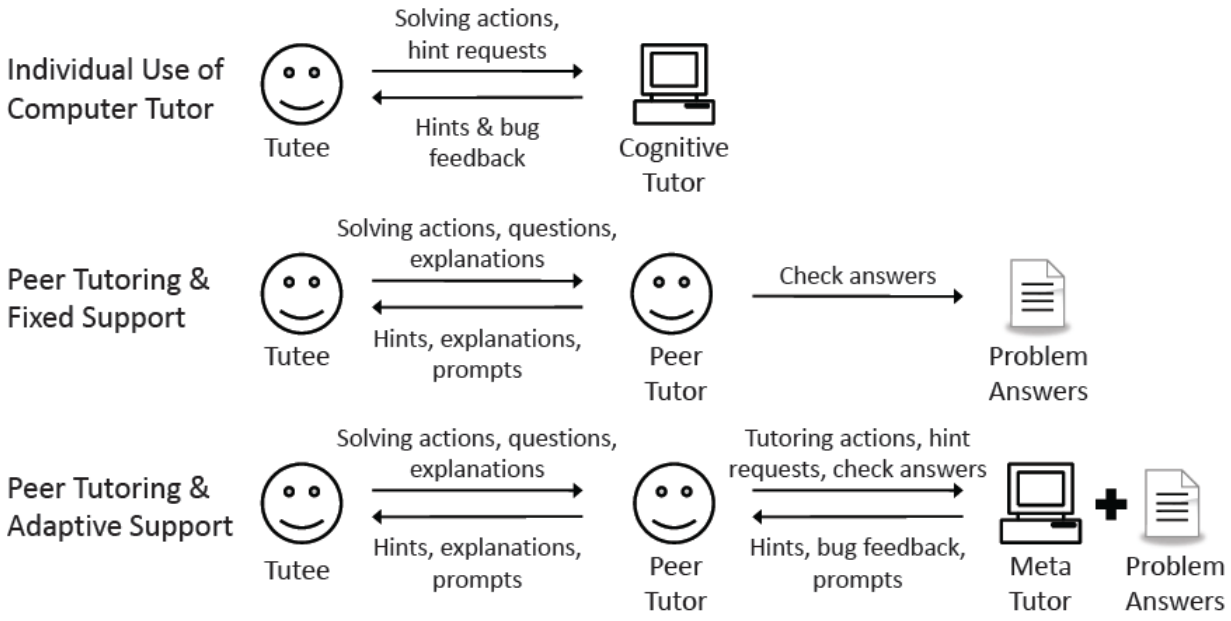

Figure 7. Three tutoring conditions

4.2 Implementation of study conditions with the CTRL architecture

In this subsection, we first discuss the high-level structure of our implementation of the three conditions, and then describe in detail how each component was implemented. All conditions were implemented as instantiations of our CTRL architecture, with a mixture of custom-implemented components and some components that were originally part of the CTA.

The *adaptive peer tutoring condition* called for two tool components: one which displays the peer tutor's interface and one which displays the peer tutee's interface. A translator component was included to echo actions from one tool to the other tool. Two tutor components were

necessary: a cognitive/domain tutor component to evaluate the peer tutee's problem-solving actions, and a meta-tutor component to evaluate the peer tutor's collaborative actions. There is a learner management component, a research management component, and a control module. Components communicate using the message protocol described in the architecture. All peer tutee solver actions, peer tutor correction actions, peer tutor skill ranking actions, and student chat actions are logged as tool messages. All cognitive and meta tutor feedback and hints are logged as tutor messages. See the left hand side of Figure 8 for a diagrammatic representation of the message-passing logic in the adaptive support condition (all interactions occur via the mediator). In this configuration, when the peer tutee takes an action, the echoing translator sends the action to the peer tutor's screen. In addition, the cognitive tutor evaluates the action, and sends the evaluation to the meta-tutor. All these interactions occur via the mediator. When the peer tutor takes an action, it is sent to the echo translator, which echoes the action onto the peer tutee's screen, and to the meta-tutor, which compares the peer tutor evaluation to the cognitive tutor evaluation. If responses do not match up, the meta-tutor sends feedback to the peer tutor. The peer tutor can also request a hint from the meta-tutor, which has stored the cognitive tutor hint for that step, and delivers it to the peer tutor. As the logic of which components are involved in the session and how components communicate exists in the control component, it is simple to use the control component to implement the relevant comparison conditions. The *fixed peer tutoring condition* includes the two tools displaying the peer tutor's interface and the peer tutee's interface, and the translator component. The *individual use condition* includes the original CTA tool and tutor components: a tool similar to the peer tutee's tool and the cognitive tutor. The right hand side of Figure 8 shows the message passing logic for these two comparison conditions. Three session types were created that corresponded to the three condition, so it was very simple to switch from one condition to another. All three conditions were fully implemented in Java. In the remainder of this section, we discuss the components involved in our implementation in more detail. In some cases, the implementation of the condition components matched the architecture exactly. However, in other cases, we will discuss instances where the implementation process lead to insights that contributed to the design of the current version of the architecture.

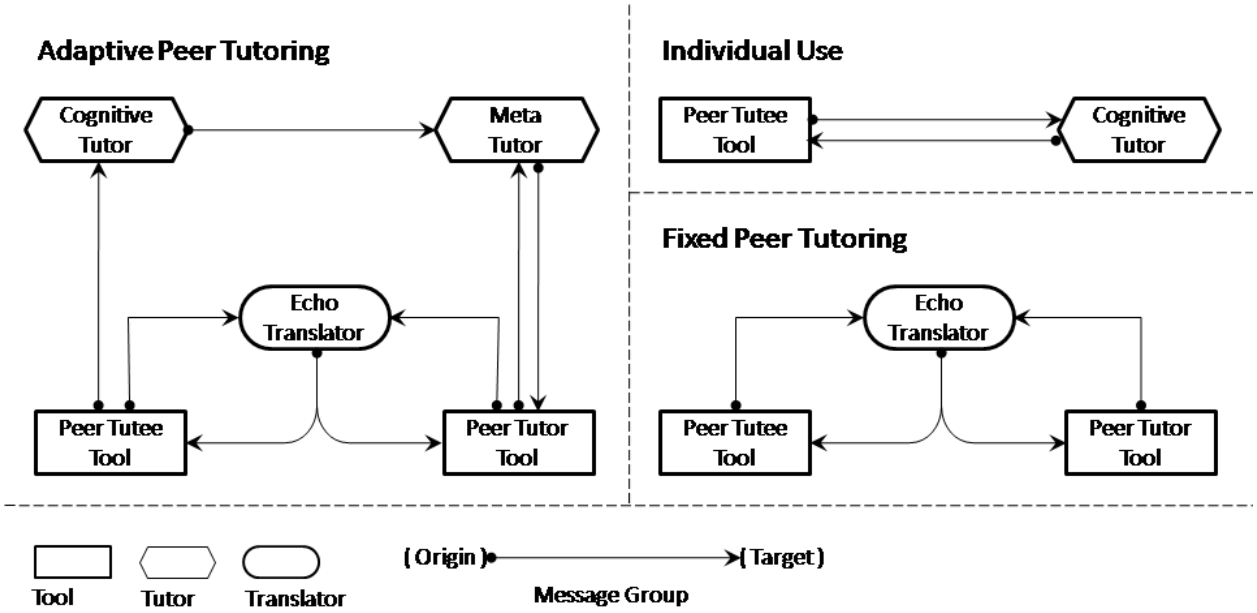

Figure 8. Session type representation for all three conditions.

The *control module*, which facilitated the flexible integration of the components necessary to form the three conditions, was composed of a *mediator* and a *session manager*. The components involved were defined in the same manner as in the current architecture, where all the components involved in a session and their component type are enumerated. However, instead of the message passing logic being defined in a rule-based manner, it was defined in the form of several “message groups”, each comprising an originating component, a target component, and a priority (represented pictorially in Figure 8).Message groups served as a template for automatically authoring simple rules. Upon receiving a message from a component, the mediator would match the component to all message groups that have that component as an origin, and then send the message to the targets in each relevant message group. In the case of messages sent to non-tool components, the control module then waits for a response from all the components that have received messages, before sending the messages out in the order of the specified priority. We intended to implement more complexity into the “message groups”, but we soon realized the limitations of the format for anything more complex than adding action specifications to the group statements, and consequently decided to go with a rule-based format in the future. Another change that we made for the final version of the architecture was that initially translator components were classified as tutor components within the mediator, but because translator components served a different purpose than tutor components, we decided to create the third component type. We also arrived at the logging format in the architecture after several iterations. Initially the messages we were logging followed the tool and tutor message format, but were missing context that had to be inserted using post-hoc code (for example, the particular student performing an action wasn’t recorded with every message). For the iteration of the architecture described in Section 3, we put the context in during the first logging pass. Like the mediator, the session manager had access to all the components involved in a session. When a session was created, the session manager would retrieve the relevant curriculum and student information, and then broadcast a launch message to all involved components, and when a

session was ended, the session manager would broadcast a quit message to all involved components. Finally, the session manager would receive a message whenever a student was done with a problem, broadcast a change problem message to all involved components, and update the student information.

The *tool components* were implemented based on the equation solver tool already found in the CTA. Although the CTA was intended to be implemented in line with Ritter & Koedinger's (1996) clean separation between tools and tutors, development constraints lead its current state to evolve from this ideal. The tool was dependent on the tutor for launching, and all the equation manipulation logic required for updating the equation after a user problem-solving step was located within the tutor. Therefore, our first step to being able to use the CTA tool components was refactoring them so that the tool functionality was completely separate from the tutor functionality. Because this process entailed learning about and working with existing code, it is important to note that it was time-consuming, and the refactored product is not as cleanly implemented as it may have been had we started from scratch. The tool components were then further modified to create the two student interfaces. The *peer tutee's interface* was the same as the regular tool interface, but with an enlarged Skillometer, an instant messaging window incorporated, and the hint button removed. The *peer tutor interface* was derived from the regular tool interface, but we removed the menus that were used to manipulate the equations and the done button. Instead, we added widgets so that the peer tutor could evaluate the steps and raise and lower the values of the skill bars. All components were implemented in Java swing, just like the original CTA interface. Although at the time of the initial implementation chat messages were not routed through the mediator, currently chat messages are being routed through the mediator, as specified by the architecture.

The *translator component* was a custom-made component designed to facilitate collaboration between two users. This component functions by receiving all processUserAction messages and converting them into corresponding scriptAction messages before sending them back to the tool components through the mediator. The translator only deals with semantic events, so the shared solver workspace is not a "what you see is what I see" interface. This decision was made to allow the peer tutee space to work without interference from the peer tutor. The CTA is set up so that the tool needs permission from a tutor to effect certain actions. Because we want the peer tutee's interaction to be less restricted than in typical use of the cognitive tutor, the translator automatically grants that permission. The translator is constructed based on the functionality of the CTA tools, and is therefore not an all-purpose component for facilitating collaboration (which, given the goal of working with existing closed components, would likely not be possible).

As mentioned above, we implemented two tutor components in the adaptive peer tutoring condition, one existing CTA component (domain tutor) and one custom-made component (meta-tutor). The domain tutor component was taken directly from the refactored CTA, without any further modifications. The meta-tutor was built fully from scratch. It consisted of an expert model based on a simple bug rule:

IF a student has taken step *x*
AND the cognitive tutor response to *x* is *a*
AND the cognitive tutor feedback message is *m*
AND the peer tutor response to *x* is *b*
AND *a* is not equal to *b*
THEN send feedback to the peer tutor using *x,a,b,m*

When this bug rule fires, the tutoring model considers the type of problem step and peer tutor response in choosing from a fixed set of collaboration-oriented meta-feedback, and then, if the domain tutor has responded with textual feedback, appends the cognitive tutor message to the feedback message. The tutoring model then sends a processTutorResponse message to highlight the problem step on the peer tutor's screen and to present the feedback to the peer tutor. Both the domain tutor and peer tutor have to have responded to the step before the rule can fire, and thus the model is not forcing the peer tutor to respond to every single tutee step. Hint requests from the peer tutor were in a similar manner:

IF a student has requested a hint
AND the next correct step is *x*
AND the cognitive tutor hint for step *x* is *y*
THEN send feedback to the peer tutor using *x* and *y*

The meta-tutor is domain-independent, and thus could be effective in combination with any intelligent tutor, as long as a translator existed to translate the intelligent tutor messages into an appropriate message format.

In the architecture, we purposefully did not specify how to pass messages remotely or how to implement a client-server framework so that multiple people could collaborate at once. Specifying such a framework is outside the scope of the architecture and might depend in part on the conditions of the classrooms in which the collaboration is being implemented (some classrooms do not allow web-based delivery, for example). Within a single collaborative session, the session manager handles launching, quitting, and navigating between problems, while the mediator handles the within-problem component exchanges. In the CTA, components had already been designed to send networked messages using TCP/IP sockets, so this is the protocol we used within the mediator to send the low-level remote messages. High-level responsibility for managing sessions was not fully factored, so we used Java RMI to make the remote message calls for accomplishing these functions. We also used RMI to implement a client-server setup for running multiple tutoring sessions at once. Once two clients that were part of the same session had connected to the server, both the session manager and mediator were started on the server, and the session type related to the user login was retrieved. All other components (tools, tutors, and translators) were run on client machines.

4.3 Experimental Study

After implementing the adaptive support condition (adaptive peer tutoring), the close comparison condition (fixed peer tutoring), and the far comparison condition (individual use), we compared the three conditions in a controlled study in a classroom (see Figure 7). We expected that both peer tutoring conditions would learn more than the individual use condition, because of the additional depth of elaboration and learning afforded through student collaboration. Furthermore we expected that the adaptive peer tutoring condition should be better than the fixed peer tutoring condition, because the adaptive support provided by the intelligent system would meet the peer tutors need for support better. This should in turn enable the peer tutor to provide better support to the peer tutee, improving the learning of both students.

Participants

Participants were 62 high-school students from five second-year algebra classes at a vocational high school in the United States, taught by the same teacher. Students spent half the day at this high school taking technical subjects (e.g., nursing or electronics) and math. The other half of the day was spent at their "home school" learning other conventional subjects. The high-school used the individual version of the CTA as part of regular classroom practice. The unit used in the study was a review unit for the students, but one that they had generally (based on the assessment of the classroom teacher) not yet mastered.

Students from each class were randomly assigned to one of the three conditions. Eleven students were excluded from analysis because either they or their partner was absent during a collaborative part of the intervention, and they could not be re-paired with another student. Another 12 participants did not take the delayed posttest, but were included for all other analyses. The total number of participants included in the analysis was thus 51 for the pre- and posttest(17 students in the adaptive peer tutoring condition, 14 students in the fixed peer tutoring condition, and 20 students in the individual use condition), and 39 students for the delayed posttest (11 in the adaptive peer tutoring condition, 10 fixed peer tutoring condition, and 18 in the individual use condition).

Experimental Procedure

The study took place over the course of three weeks. Students were given a 15 minute pretest on the Monday or Tuesday of the first week, depending on their class schedules. The intervention then took place on two days, over two 70 minute class periods. The first intervention day was on the Thursday or Friday of the first week, the second was on Thursday or Friday of the following week. On both intervention days, students in the peer tutoring conditions spent half the period in the preparation phase, took a brief intermediate test, and spent the remaining classroom time taking turns tutoring each other in the collaboration phase. Students in the individual use condition used the CTA during the preparation phase, took the intermediate test, and then continued to use the CTA alone during the collaboration phase. The week after the intervention, students were given a 15 minute posttest. Two weeks later, students were given a 15 minute delayed posttest to assess their long-term retention.

Measures

To assess students' individual learning we used counterbalanced pre-, post-, and delayed posttests, each containing 8 questions at 3 levels of difficulty. We expected that most students would not be able to solve all questions in the time given, and gave instructions to attempt as many as they could. The tests were administered on paper. For process data, we logged all tutor actions, tutee actions, and intelligent tutor responses. In other words, the interface actions in both tool components were captured (e.g., the tutee subtracts m from both sides) and dialog actions (e.g., the tutee asks "what do I do?"). The log data allowed us to extract process variables such as the incorrect attempts made by students, the help accessed by the peer tutor, the help communicated by the peer tutor, and the total number of problems completed.

Results & Discussion

Here, we look at how the implementation of the three experimental conditions in line with our architectural goals helped us to gain insight into the learning effects of adaptive support for peer tutoring. First, we demonstrate that the multiple streams of interaction data collected by

the system, in connection with outcome measures, provided us with insight into the peer tutoring process. Next, we examine how the integration of the domain and collaboration/meta support may have affected the peer tutor's behavior. Finally, we discuss how our comparison conditions provide us with more insight than the experimental condition would have alone.

To demonstrate the *effectiveness of collecting rich log data*, we focus on one particular result in the adaptive peer tutoring condition: relating student impasses (computed using incorrect attempts at a problem-solving step and incorrect attempts to move to the next problem) to the learning gains between pretest and delayed test (computed using a normalized gain score). One might expect that the more mistakes a tutee makes, the less they would gain between the pretest and delayed posttest, because if they made many mistakes throughout the intervention they probably have not mastered the material. Sure enough, in the adaptive condition, the total number of incorrect problem solving attempts on the part of the tutee were negatively correlated with tutee learning (r = -.614, p = .044), as were the total number of incorrect attempts to move to the next problem (r = -.591, p = .056). One might also expect that the more mistakes a tutee makes, the worse their tutor will do on the delayed posttest, as large numbers of tutee mistakes may indicate that the tutor lacks the understanding to successfully help their tutee through the problem. However, surprisingly, tutor learning was *positively* correlated with tutee incorrect problem solving attempts (r = .428, p = .190) and tutee incorrect done attempts (r = .463, p = .151). It appears that much of this learning results from the peer tutor actively processing the tutee errors, as was similarly demonstrated by studies on learning from erroneous worked examples (Große & Renkl, 2007). This insight would not have appeared had we not collected both problem-solving and outcome data.

The relationship between the chat and problem-solving data also provides us with information that would not have been available had we only had one source of interaction data. Table 4 displays a student interaction immediately after the peer tutee has taken an incorrect done action and the peer tutor has incorrectly agreed and has therefore received a feedback message that the problem was not in fact done. At this point, the equation the students are working on is "$t = f / (1 - .75)$", with the goal being to solve for t. The students must realize here that they need to get rid of the decimal in the denominator to achieve the answer "$t = 4f$". The entire exchange in Table 4 took 10 minutes. If we were to look only at the left hand column of Table 4, depicting the student talk, it might appear that productive behaviors are not occurring at all: the tutor is simply giving the tutee didactic instructions for how to proceed. Looking at the problem-solving actions does appear to confirm a lack of effort on the part of the tutee. We see that the tutee is essentially taking a trial and error approach to completing the problem, executing both tutor suggestions and other viable options, and then attempting to finish the problem by clicking done. However, throughout this interaction the tutor consults the problem actions after every tutee action, suggesting that the tutor is engaged in comparing the student answer to the ideal worked example. We also notice that this tutor does not make use of the adaptive feedback provided, which may have helped him in making the comparison. Further, we can see at a glance which actions are correct or incorrect, and when feedback was given. Using these multiple streams of data, we can better understand that tutors may be benefiting from student errors by being encouraged to compare the errors to a correct problem solution. In fact, this tutor had a gain score on the delayed test of .33, suggesting that some of this active processing was beneficial, but also that there was more room for improvement.

Table 4. Student interaction immediately following an impasse.

Chat Actions	Problem-Solving Actions	Computer Response
	Tutor: checks answers	
Tutee: yeah i donno what to do after that step *Tutor*: simplify fractions i think		
	Tutee: simplify fractions	Incorrect (cognitive)
	Tutee: undoes simplify fractions	
	Tutor: checks answers	
Tutee: I did that		
	Tutee: combine like terms	Correct (cognitive)
	Tutee: clicks done	Incorrect (cognitive)
	Tutor: agrees done	Incorrect (meta)
	Tutor: checks answers	
	Tutee: clicks done	Incorrect (cognitive)
	Tutor: agrees done	Incorrect (meta)
Tutor: multiply by 4		
	Tutor: checks answers	
Tutee: both sides		
Tutor: no		
	Tutee: performs multiplication	Incorrect (cognitive)
	Tutee: undoes perform multiplication	
	Tutor: checks answers	
Tutor: I mean yes		
	Tutee: multiplies both sides by 4	Incorrect (cognitive)

The addition of adaptive feedback that incorporates both domain support and collaborative/meta support might provide insight into how providing adaptive feedback affects the peer tutor's behavior. In our adaptive condition, peer tutors were given domain feedback about the peer tutee's actions and then instructions to communicate the feedback to the tutee. They also had access to the problem answers as they were tutoring. To make a fair comparison, we look only at the 9 students who chose to use all forms of assistance when in the role of the peer tutor. We examine which assistance they used and whether they communicated the assistance or not. As evident from Table 5, students accessed far more fixed assistance than adaptive assistance. However, they were relatively more likely to communicate the adaptive assistance that they received than the fixed assistance: they communicated half the adaptive assistance they received, but only 39% of the fixed assistance they received. This may be due to the prompt to communicate embedded in the adaptive assistance. This observation is particularly interesting given the differences in the effects of the two types of assistance received by the tutor on the gains of the student in the peer tutee role. As might be expected, communicating adaptive assistance was positively correlated with tutee learning, while failing to communicate adaptive assistance was negatively correlated with tutee learning (represented by the third column in Table 5). Perhaps surprisingly, communicating domain feedback to the tutee after accessing fixed support (i.e. checking the answers to the problem) was negatively correlated with tutee learning.

Table 5. Amounts of adaptive and fixed assistance communicated and not communicated. Effects of communicating the assistance on tutee learning. *R* represents the correlation with the tutee learning, and *p* is the significance of r.

	Adaptive Assistance				**Fixed Assistance**			
	M	SD	R	p	M	SD	r	p
Assistance communicated	1.44	1.81	.786	.115	3.56	3.84	-.925	.024
Assistance not communicated	1.44	2.01	-.803	.102	5.67	6.22	-.331	.587

We can retrieve from our log data examples from each of the three different types of assistance used that correlates with tutee learning to illustrate what may be occurring. The following is an example of the peer tutor receiving feedback on a step, and not communicating it to his or her partner: The peer tutor has marked a step right. He receives a feedback message telling him that the step was actually wrong and giving him a hint on the step. At this point, the peer tutee says: "that doesn't look right, im sorry I suck at math lol", and then "k, nevermind." The peer tutor does not respond. Then the peer tutee clicks done, the peer tutor agrees, and the peer tutor is given another feedback message saying the problem is not done. This message is not communicated to the tutee either. Given such lack of communication, not only are tutees not getting the assistance needed, but they are getting misleading feedback. To the tutee, it appears as if the steps are correct, even if they are not.

This example can be compared to an example where the feedback the peer tutor received was communicated. When a different tutor received a message saying that their partner had made a calculation error (after marking an incorrect step right), an extended dialogue ensued as follows:

Tutor: undo it
Tutee: why? U marked it right….?
Tutor: The step is right but it said you made a typing error when you factored
Tutee: in which step?
Tutor: the first
Tutee: so u want me to undo it or is it right?
Tutee: k
Tutor: undo it

Not only did the tutor communicate what was incorrect about the current problem solution, the two students also together cleared up a misunderstanding about which aspect of the step was incorrect. This type of dialogue may be less common when the fixed assistance is communicated to the tutee.

In the next example, where fixed assistance is provided to the tutee, after the tutee takes an incorrect step dividing both sides by $q + r$, the peer tutor checks the answers and then says, "divid both sides by $q + r$." The tutee then promptly undoes his last step and performs the correct step. Here, it is likely that the tutor instruction was not beneficial, because no explanation was provided for why the first step was wrong and the second step was right, and the tutee did not have to identify or reflect on his error. The final scenario involved the tutor accessing the fixed assistance without communicating it. In this case, the tutee is typically engaged in completing problem steps as the tutor is using the answers to check the correctness of the steps. It is unsurprising that this case had little effect on partner gains, as the tutor's intervention was neither beneficial nor disruptive behavior.

As our research platform enabled us to develop two comparison conditions in addition to the adaptive support condition, we were able to compare adaptive support for peer tutoring to fixed support for peer tutoring and to individual learning. If we had looked at the adaptive peer tutoring condition independently of the comparison conditions (as many ACLS evaluations have done so far), we would have found that the mean learning gains in the adaptive condition appear satisfactorily high, with an improvement of 34% on test scores between the pretest and the delayed posttest. However, comparing the learning improvement across all three conditions, we see that the adaptive condition is not much different from the fixed support condition (45% improvement) or the individual use condition (40% improvement). Then, even though the learning gains across the three conditions are equal, we can examine the different paths students took to learning across the three conditions. For example, the number of problems completed per hour in the individual condition (M=47.0, SD = 30.2) were much higher than the number of problems completed per hour in the fixed support condition (M=13.3, SD=7.71) and the adaptive support condition (M=17.7, SD=5.69). A logical hypothesis may be that students in the individual conditions students learned by solving many problems quickly but shallowly, whereas in the collaborative conditions students learned by solving fewer problems slowly but deeply. It would not be possible to place the effects of the adaptive support in context without the results of the comparison conditions.

4.4 Summary

We designed a collaborative peer tutoring script and adaptive domain support for the peer tutoring, implemented the adaptive support condition and two comparison conditions using the CTRL architecture, and conducted a controlled classroom study comparing the three conditions. As a result, we gained valuable insights on the effects of providing adaptive support to peer tutoring. We were able to use the combination of process data and outcome data to learn that the more impasses faced by tutees, the more their tutors showed delayed posttest gains. We used multiple streams of the process data to analyze why that might be the case. We looked at how the adaptivity of the support related to whether the assistance was communicated or not, and investigated how those two factors related to the tutee's learning gains. Finally, were able to put the results on the adaptive condition in context by comparing it to the other two conditions. We realized that even if the learning gains were similar in all conditions the paths to learning might be different. In conclusion, implementing the experimental conditions of the learning system with the CTRL architecture facilitated the learning sciences research that we conducted.

5 Conclusions

We have outlined a framework that allows us to compare the effects of different types of adaptive assistance on student collaborative process, which can then be linked to their learning outcomes. Our goal was to build an architecture that can help researchers to contribute to the learning sciences by capturing student collaborative interactions and using them to evaluate the effects of interventions, both after the fact and as real-time input to an adaptive feedback system. The architecture enables researchers to integrate different types of adaptive support, particularly in terms of leveraging domain-specific models as input to domain-general components in order

to create more complex tutoring functionality. Additionally, the architecture helps researchers to implement comparison conditions by making it easier vary single aspects of the adaptive intervention by removing tool or tutor components from a system. We evaluated the architecture by first designing adaptive and fixed support for a peer tutoring script, then instantiating the architecture using those two scenarios and an individual tutoring scenario. Implementation was accomplished by combining pre-existing components from the Cognitive Tutor Algebra (CTA) with custom-built components. The three conditions were compared in a large-scale study in the classroom that already incorporated the CTA as part of regular classroom practice. The results helped us to contribute to learning sciences research in peer tutoring, in particular by illuminating the relationship between tutee impasses and tutor learning, giving insights into the different effects of communicating adaptive and fixed assistance, and revealing the different paths students take to learning in individual and collaborative conditions.

We see one of the main contributions of our work as the development of an architecture that facilitates the integration of pre-existing components and custom-built components, with a particular focus on tutoring components. At first glance, it seems like it might not be a good idea to rely heavily on existing tutoring systems for components, in that it may be necessary to refactor the components to fit into the framework or to deal with legacy code that it may be difficult to appropriate for new purposes. However, in our three implemented conditions, we managed to leverage CTA logging protocols, interface components, and cognitive models, which would have been much more time-consuming to reconstruct from scratch. These components made it possible to develop a classroom-functional adaptive collaborative learning system, which is currently a rarity. Another concern with relying too much on existing components is that it might overly constrain the design of adaptive support interventions. It is true that considering the full design space of adaptive collaborative learning support, our system did not depart very much from the current functionality of the CTA. It substitutes peer tutoring for cognitive tutoring and collaborative domain support for individual domain support, but we did not explore collaborative scenarios that do not involve tutoring or forms of collaborative support other than collaborative domain support. It seemed like remaining close to the intelligent tutoring system was the first natural step, and further extensions belong in future work; particularly since there is much more to be done within the confines of existing CTA components (e.g., leveraging the student modeling to help the peer tutor figure out what the student knows and doesn't know). As we develop more components, they will form a basis to help construct more general collaborative scenarios.

The other main contribution of our architecture is how it makes it easier to implement and compare conditions, by keeping all the integration logic out of the main tutoring components and in a central mediator. In our evaluation, we compared three very different scenarios: a computer-student intelligent tutoring condition, a student-student peer tutoring condition, and a computer-student-student adaptive collaborative learning condition. Traditionally, these scenarios would have been implemented in very different manners, rather than all fitting in to the same framework. Furthermore, adding a new scenario took no more than *X* lines of code once the components had been implemented. One limitation of the architecture, however, is that currently only simple integration scenarios are accommodated; the architecture has not been tested with more complex configurations (e.g., the mediator has not yet had to deal with two competing bits of feedback). As the conditions we attempt to implement evolve to become more complex, so will the framework.

CTRL is one of the few adaptive collaborative learning support architectures to yield an implementation of an adaptive support system and a controlled evaluation of the system in a classroom, where an adaptive peer tutoring condition was compared to two comparison conditions that were also implemented using CTRL. This evaluation was to accomplish the learning sciences goal of better understanding the effects of adaptive support on peer tutoring, and the way CTRL was designed helped us to achieve that goal. The integrated log of verbal interaction and problem-solving data helped us both to link student impasses to posttest scores and better understand what might be occurring as students reach impasses. One downside to analyzing rich data is that it is more time-consuming than examining a single stream of data. In some ways, CTRL allows us to streamline this process by collecting and integrating all data automatically, and by encouraging researchers to store semantic-level representations of what is occurring rather than low-level interface actions. Further, even though multiple streams of data is collected by CTRL, it is still reasonable to focus only on analyzing a subset of the data; essentially, it is better to make more data available than less. Another potential disadvantage of CTRL is that our focus has been on text-based communication rather than the video or audio conferencing that is a better simulation of face-to-face interaction. Using text-based communication allows us to more easily automatically analyze the content of the communication, but removes input channels, and may be less natural for students. However, the students that we use tend to be very comfortable with instant messaging applications and have found ways to compensate for the lack of visual input, so in our experience the tradeoff is justified. Another research avenue facilitated by CTRL is the evaluation of complex adaptive support based on different types of collaborative models. In our study, we combined a domain model of the tutee's problem-solving performance with a collaborative model of the peer tutor's correction actions, and delivered feedback that included both a prompt to collaborate and a domain hint. We were able to closely examine the effects of this feedback, in particular with respect to the effects of the peer tutor receiving and communicating the feedback. It is true that other ACLS systems have also delivered multiple types of adaptive feedback, and in some cases the feedback that they have delivered has been more complex than ours. However, even though the current system we have implemented is not very complex, our hope is that our architecture will allow us to incrementally increase the complexity of the system and vary the types of adaptive support provided in a way that other ACLS architectures have not. One more research goal facilitated by CTRL was the more efficient implementation of comparison conditions by easily removing components from the definition of a collaborative session. Because of this capability, we were able to place our adaptive support results in the context of both fixed support and individual work with a tutoring system. Using this type of integration framework to create comparison conditions means that the conditions that can be created depend on the way components are divided, and some finer-grained comparisons might require more effort to implement (e.g., comparing two different feedback policies). Although we cannot cover the full spectrum of comparison conditions that might be desired, we believe that the space of conditions that can be implemented with our architecture makes conducting research using the architecture much more efficient.

CTRL, the collaborative tutoring research lab, is an initial first step toward supporting research into complex forms of adaptive assistance toward collaborative learning. Not many ACLS architectures move from the design phase to the implementation phase, and even fewer form the basis for an evaluation of the effects of the adaptive support. Even architectures that do reach the evaluation stage are not often used in classrooms or in controlled studies. It is our hope

that the structure of CTRL, and in particular its integration framework, facilitated a more efficient implementation by leveraging domain-specific models and a more controlled evaluation by allowing the construction of control conditions using the same components and the same architecture. Further experimentation under this paradigm will allow us to increase the scope of implemented scenarios and the complexity of our support.

References

Aleven, V., & Koedinger, K. R. (2002). An effective meta-cognitive strategy: learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science*, 26(2), 147-179.

Aleven, V., McLaren, B., Roll, I., & Koedinger, K. (2006). Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor. International Journal of Artificial Intelligence and Education, 16, 101-128.

Aleven, V., Roll, I., McLaren, B., Ryu, E.J., & Koedinger, K. R. (2005). An architecture to combine meta-cognitive and cognitive tutoring: Pilot testing the Help Tutor. In C. K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.), *Proceedings of the 12th International Conference on Artificial Intelligence in Education, AIED 2005* (pp. 17-24). Amsterdam, IOS Press.

Anderson, J. R., & Pelletier, R. (1991). A development system for model–tracing tutors. In *Proceedings of the International Conference of the Learning Sciences* (pp. 1–8). Evanston, IL.

Azevedo, R. (2005b). Computer environments as metacognitive tools for enhancing learning. *Educational Psychologist*, 40(4), 193–197.

Baghaei, N., Mitrovic, A., Irwin, W (2007). Supporting collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams.

Baker, R.S.J.d., Corbett, A.T., Koedinger, K.R., Evenson, E., Roll, I., Wagner, A.Z., Naim, M., Raspat, J., Baker, D.J., Beck, J. (2006) Adapting to When Students Game an Intelligent Tutoring System. Proceedings of the 8th International Conference on Intelligent Tutoring Systems, 392-401.

Beck, J.E., Mostow, J., & Bey, J. (2004). Can automated questions scaffold children's reading comprehension? The 7th International Conference on Intelligent Tutoring Systems.

Biswas, G., Leelawong, K., Schwartz, D., Vye, N. & The Teachable Agents Group at Vanderbilt (2005). Learning By Teaching: A New Agent Paradigm for Educational Software, Applied Artificial Intelligence, vol. 19, (pp. 363-392).

Braun, I. (2008). Promoting chemistry learning through scripted collaboration. Structural and adaptive support for collaboration in a computer-supported learning environment. Unpublished Diploma Thesis. Institute of Psychology, University of Freiburg, Germany.

Constantino-Gonzalez, M. A., Suthers, D., & Escamilla de los Santos, J. (2003). Coaching web-based collaborative learning based on problem solution differences and participation. Artificial Intelligence in Education, 13(2–4), 263–299.

Del Solato, T., & du Boulay, B. (1995). Formalization and implementation of motivational tactics in tutoring systems. Journal of Artificial Intelligence in Education, 6(4), 337–378.

Dillenbourg, P., Baker, M., Blaye, A., & O'Malley, C. (1995) The evolution of research on collaborative learning. In E. Spada & P. Reiman (Eds) Learning in Humans and Machine: Towards an interdisciplinary learning science. (Pp. 189-211). Oxford: Elsevier.

Dillenbourg, P., & Hong, F. The mechanics of CSCL macro scripts. In the *International Journal of Computer-Supported Collaborative learning, 3*(1), March 2008, pp. 5-23.

Dillenbourg, P. (2002). Over-scripting CSCL: The risk of blending collaborative learning with instructional design. In Kirschner, P. A. (Ed) Three worlds of CSCL: Can we support CSCL?, Heerlen: Open Universiteit Nederland, 61-91.

Fantuzzo, J. W., King, J., & Heller, L. (1992). Effects of reciprocal peer tutoring on mathematics and school adjustment: A componential analysis. *Journal of Educational Psychology,* 84, 331-339.

Fantuzzo, J. W., Riggio, R. E., Connelly, S., & Dimeff, L. A. (1989). Effects of reciprocal peer tutoring on academic achievement and psychological adjustment: A component analysis. *Journal of Educational Psychology*. 81(2), 173-177.

Fischer, F., Kollar, I., Mandl, H., & Haake, J.M. (Eds.) (2007). Scripting computer-supported collaborative learning – cognitive, computational, and educational perspectives. New York: Springer.

Fuchs, L., Fuchs, D., Hamlett, C., Phillips, N., Karns, K., & Dutka, S. (1997). Enhancing students' helping behavior during peer-mediated instruction with conceptual mathematical explanations. *The Elementary School Journal*, 97(3), 223-249.

Genesereth, M. R. 1997, An agent-based framework for interoperability, In J. M. Bradshaw (Ed), *Software Agents* (317-345), Menlo Park, California: AAAI Press/MIT Press.

Große, C. S. & Renkl, A. Finding and fixing errors in worked examples: Can this foster learning outcomes? *Learning & Instruction*, 17, 612-634 (2007)

Gweon, G., Rosé, C., Carey, R. & Zaiss, Z. (2006). Providing Support for Adaptive Scripting in an On-Line Collaborative Learning Environment. Presented at *CHI 2006*.

Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? Educational Psychology Review, 16, 235–266.

Hoppe, H. U. 1995, The use of multiple student modeling to parameterize group learning, In J. Greer (Ed), Proceedings of AI-ED 95, Washington D.C., USA.

Hoppe, H. U. & Gaßner, K. (2002). Integrating collaborative concept mapping tools with group memory and retrieval functions. In G. Stahl (Eds.), Computer Support for Collaborative Learning: Foundations for a CSCL Community . Boulder, USA: distrib. Lawrence Erlbaum.

Israel, J. and Aiken, R. "*Supporting Collaborative Learning with an Intelligent Web-based System*", International Journal of Artificial Intelligence and Education (IJAIED), 2007.

Jordan, P., Hall, B., Ringenberg, M., Cui, Y., Rosé, C. P. (2007). Tools for Authoring a Dialogue Agent that Participates in Learning Studies, Proceedings of AIED 2007.

King, A., Staffieri, A., & Adelgais, A. (1998). Mutual peer tutoring: Effects of structuring tutorial interaction to scaffold peer learning. *Journal of Educational Psychology*, 90, 134-152.

Koedinger, K. R., & Aleven V. (2007). Exploring the assistance dilemma in experiments with Cognitive Tutors. *Educational Psychology Review*, 19(3), 239-264.

Koedinger, K., Anderson, J., Hadley, W., & Mark, M. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43. (1997)

Koedinger, K., & Corbett, A. *Cognitive Tutors: Technology bringing learning science to the classroom.* In K. Sawyer (Ed.), The Cambridge Handbook of the Learning Sciences, Cambridge University Press. 2006. Pages 61-78.

Koedinger, K. R., Suthers, D., & Forbus, K.D. (1999). Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10, 292-313.

Krueger, C.W. (1992). Software Reuse. ACM Computing Surveys, 24 (2), 131-183.

Kumar, R., Rosé, C. P., Wang, Y. C., Joshi, M., Robinson, A. (2007). Tutorial Dialogue as Adaptive Collaborative Learning Support, *Proceedings of AIED 2007*.

McNamara, D. S., O'Reilly, T., Rowe, M., Boonthum, C., & Levinstein, I. B. (in press). iSTART: A web-based tutor that teaches self-explanation and metacognitive reading strategies. In D.S. McNamara (Ed.), *Reading comprehension strategies: Theories, interventions, and technologies*. Mahwah, NJ: Erlbaum.

Medway, F. & Baron, R. Locus of control and tutors' instructional style. *Contemporary Educational Psychology*, 2, 298-310 (1997).

Meier, A., Spada, H., Rummel, N.: A Rating Scheme for Assessing the Quality of Computer-Supported Collaboration Processes. International Journal of Computer-Supported Collaborative Learning 2, 63–86 (2007)

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

Mühlenbrock, M., Tewissen, F. & Hoppe, H. U. (1998). A framework system for intelligent support in open distributed learning environments. International Journal of Artificial Intelligence in Education, 9, 256-274.

Pinkwart, N. (2003). A Plug-In Architecture for Graph Based Collaborative Modeling Systems. In U. Hoppe, F. Verdejo & J. Kay (eds.): Proc. of Artificial Intelligence in Education, Amsterdam, IOS Press.

Ritter, S., Blessing, S. B., & Hadley, W. S. (2002). SBIR Phase I Final Report 2002. Department of Education. Department of Education RFP ED: 84-305S.

Rosatelli, M., & Self, J. (2004). A collaborative case study system for distance learning. International Journal of Artificial Intelligence in Education, 14, 97-125.

Roschelle, J., Kaput, J., Stroup, W., & Kahn, T. M. (1998). Scaleable integration of educational software: Exploring the promise of component architectures. *Journal of Interactive Media in Education,* 1998(6).

Roscoe, R. D. & Chi, M. (2007a). Understanding tutor learning: Knowledge-building and knowledge-telling in peer tutors' explanations and questions. *Review of Educational Research. 77*(4). 534-574.

Roscoe, R. D. & Chi, M. (2007b). Tutor learning: The role of instructional explaining and responding to questions. *Instructional Science. 36*(4). 321-350.

Rummel, N., Diziol, D., Spada, H., & McLaren, B. (2007). Scripting Collaborative Problem Solving with the Cognitive Tutor Algebra: A way to Promote Learning in Mathematics; Paper presented at the *Conference by the European Association for Research on Learning and Instruction (EARLI 2007)* as part of symposium on 'Scripting (collaborative) learning on different social levels' by Pivi Hökkinen.
Rummel, N., & Spada, H. (2005). Learning to collaborate: An instructional approach to promoting collaborative problem-solving in computer-mediated settings. Journal of the Learning Sciences, 14, 201-241.

Rummel, N. & Weinberger, A. (2008, June). New challenges in CSCL: Towards adaptive script support. Symposium to be presented at the Internationcal Conference of the Learning Sciences, Utrecht, NL.

Saab, N., van Joolingen, W.R., & van Hout-Wolters, B.H.A.M. (2007). Supporting Communication in a Collaborative Discovery Learning Environment: the Effect of Instruction. Instructional Science, 35, 73-98.

Strijbos , J. W., Martens , R. L., & Jochems, W. M. G. (2004). Designing for interaction: six steps to designing computer-supported group-based learning, Computers & Education, v.42 n.4, p.403-424 .

Slavin, R. E. (1996). Research on cooperative learning and achievement: What we know, what we need to know. *Contemporary Educational Psychology, 21*, 43-69.

Soller, A., Martinez, A., Jermann, P., and Muehlenbrock, M. (2005).From mirroring to guiding: A review of state of the art technology for supporting collaborative learning.International Journal of AI-Ed, 15:261–290.

Soller, A. (2004). Computational modeling and analysis of knowledge sharing in collaborative distance learning. User Modeling and User-Adapted Interaction: The Journal of Personalization Research, 14 (4), 351-381.

Suebnukarn, S., Haddawy, P.: A collaborative intelligent tutoring system for medical problem-based learning. Proceedings of the 9th International Conference on Intelligent User Interfaces. (2004) 14-21

Suthers, D.D., (2001), Architectures for Computer Supported Collaborative Learning, Proc. IEEE int. Conf. On Advanced Learning Technologies, ICALT 2001, Madison, Wisconsin

Tedesco, P. (2003). MArCo: Building an artificial conflict mediator to support group planning interactions. International Journal of Artificial Intelligence in Education, 13, 117-155.

Tsovaltzi, D., Rummel, N., Pinkwart, N., Scheuer, O., Harrer, A., Braun, I., & McLaren, B. (2008). CoChemEx: Supporting conceptual chemistry learning via computer-mediated collaboration scripts. Paper to be presented at EC-TEL 2008, Maastricht (proceedings to be published in Lecture Notes in Computer Science, Springer).

VanLehn, K. (2006) The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education. 16* (3), 227-265.

VanLehn, K., Koedinger, K. R., Skogsholm, A., Nwaigwe, Adaeze, Hausmann, R.G. M., Weinstein, A., Billings, Benjamin: What's in a Step? Toward General, Abstract Representations of Tutoring System Log Data. User Modeling 2007: 455-459.

Vieira, A. C., Teixeira, L., Timóteo, A., Tedesco, P., Barros, F. A. (2004). Analyzing on-line collaborative dialogues: The OXEnTCHÊ-Chat. In J. C. Lester, R. M. Vicari, F. Paraguaçu (Eds.): The 7th International Conference on Intelligent Tutoring Systems, ITS 2004, Maceiò, Alagoas, Brazil, 315-324.

Vizcaino, A., Contreras, J., Favela, J., & Prieto, M. (2000). An adaptive, collaborative environment to develop good habits in programming. Proceedings of the 5th International

Walker, E., Rummel, N., McLaren, B., and Koedinger, K (2007). The Student becomes the Master: Integrating Peer Tutoring with Cognitive Tutoring. In the Proceedings of the Conference on Computer Supported Collaborative Learning (CSCL-07).

http://learnlab.web.cmu.edu/dtd/