User Model User-Adap Inter (2009) 19:387–431
DOI 10.1007/s11257-009-9069-1

ORIGINAL PAPER

CTRL: A research framework for providing adaptive collaborative learning support

Erin Walker · Nikol Rummel · Kenneth R. Koedinger

Received: 5 August 2008 / Accepted in revised form: 2 September 2009

Abstract There is evidence suggesting that providing adaptive assistance to collaborative interactions might be a good way of improving the effectiveness of collaborative activities. In this paper, we introduce the Collaborative Tutoring Research Lab (CTRL), a research-oriented framework for adaptive collaborative learning support that enables researchers to combine different types of adaptive support, particularly by using domain-specific models as input to domain-general components in order to create more complex tutoring functionality. Additionally, the framework allows researchers to implement comparison conditions by making it easier to vary single factors of the adaptive intervention. We evaluated CTRL by designing adaptive and fixed support for a peer tutoring setting, and instantiating the framework using those two collaborative scenarios and an individual tutoring scenario. As part of the implementation, we integrated pre-existing components from the Cognitive Tutor Algebra (CTA) with custom-built components. The three conditions were then compared in a controlled classroom study, and the results helped us to contribute to learning sciences research in peer tutoring. CTRL can be generalized to other collaborative scenarios, but the ease of implementation relates to the complexity of the existing components used. CTRL as a framework has yielded a full implementation of an adaptive support system and a controlled evaluation in the classroom.

E. Walker (✉) · K. R. Koedinger
Human-Computer Interaction Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
e-mail: erinwalk@andrew.cmu.edu

K. R. Koedinger
e-mail: koedinger@cmu.edu

N. Rummel
Institute of Psychology, University of Freiburg, Engelbergerstr. 41, 79085 Freiburg, Germany
e-mail: rummel@psychologie.uni-freiburg.de

Keywords Adaptive collaborative learning support · Intelligent collaborative learning systems · Cognitive tutoring systems · Collaboration modeling · Peer tutoring · Classroom evaluation

1 Introduction

Over the past 30 years there has been an evolution in research on how students learn by collaborating, depicted in Fig. 1 (Dillenbourg et al. 1995). In the *conditions stage,* early work compared the effects of collaborative and individual activities, 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 and relate them to outcomes (the *interactions stage*). 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 et al. 2004). Much current collaborative learning research is situated in this *fixed support stage,* which focuses on the effects of giving students *fixed* assistance, including declarative instruction on how to collaborate (e.g., Saab et al. 2007), examples of good collaboration (e.g., Rummel and Spada 2005), and collaboration scripts that provide students with designated roles and activities as they work together (e.g., Fischer et al. 2007). However, 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). On the other hand, pre-collaboration training may provide too little support for students during the actual collaboration, where students may not follow the activity as designed (e.g., Ritter et al. 2002). Adaptive support might be a better way of targeting the individual needs of students (Soller et al. 2005; Kumar et al. 2007). Therefore, 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 determine the content of the assistance given (*adaptive support stage*).

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 and Aleven 2007), where in order to discover how best to 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. Non-technological implementations of adaptive support for collaboration require an experimenter or a teacher to interact with each collaborating group (e.g., Tsovaltzi et al. 2008; Hmelo-Silver 2004; Gweon et al. 2006). This

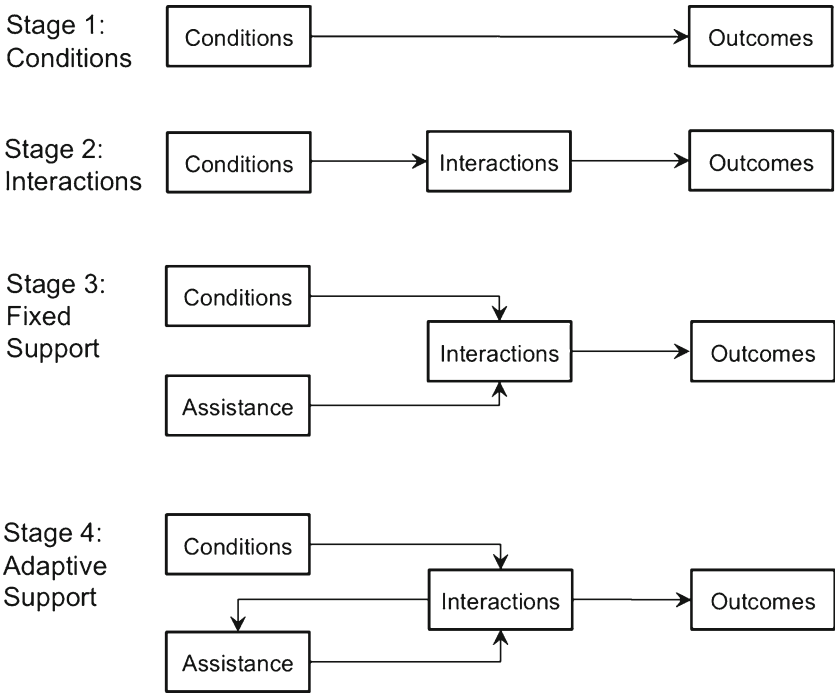

Fig. 1 Four stages of research on collaborative learning. Researchers began by examining how the collaborative setting and positive collaborative behaviors lead to learning. They then explored how to support collaboration to encourage productive interactions. More recently, the effects of adaptive support have been investigated

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 examine the effects of computer-delivered 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. Further, the effects of adaptive feedback provided by these systems on collaborative interactions and learning outcomes have rarely been evaluated in large-scale controlled studies, despite the fact that the evaluations that have been conducted 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. In general the adaptive collaborative learning systems that have been developed have been research prototypes, which mainly demonstrate how to construct such systems. This agenda for development makes it difficult to adapt them to create relevant control conditions, deploy them in classroom environments, or iterate upon them in future studies. Removing some of the technical obstacles to

implementing adaptive assistance and relevant comparison conditions may encourage the use of such systems to address educational psychology research questions.

In this paper, we introduce the Collaborative Tutoring Research Lab (CTRL), a research-oriented framework 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 CTRL, we have adopted ideas from an individual learning perspective on delivering adaptive instruction: cognitive tutors. 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 et al. 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 et al. 2004). CTRL extends the individual tutoring scenario (one student, one tutor) to a collaborative multi-tutor setting (multiple students and multiple tutors, with different roles or for different purposes). One of the strengths of our framework is that it focuses on reusability: it facilitates the addition, removal, and integration of components. In CTRL, adaptive collaborative conditions can be developed more rapidly by using existing computational models, and comparison conditions can be created by removing particular components of the adaptive system. We have used CTRL to create an adaptive support condition for a peer tutoring activity that integrates domain and collaboration assistance, and evaluated the adaptive collaboration support condition in a controlled classroom study by comparing it to a fixed collaboration support condition and an individual learning condition. All three study conditions were implemented following the framework, and the results of the study increase understanding of the effects of adaptive support on peer tutoring. In this paper, we review other adaptive collaborative learning systems in Sect. 2. In Sect. 3, we describe CTRL, and in Sect. 4, we describe the implementation and evaluation of a specific adaptive collaborative learning system using CTRL. To conclude in Sect. 5, we outline the scope of CTRL, demonstrating how it can be used to implement other adaptive collaboration scripts that involve more participants, more balanced collaborative roles, and more complex adaptive tutoring.

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 et al. (2005) in their review of collaboration support systems. Coaching systems help students who are 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. Further, we examine how cognitive tutor principles and architectures for individual learning might contribute to our goal.

2.1 Student interactions in adaptive collaborative learning support systems

ACLS systems 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 (see Dillenbourg and Hong 2008, for further discussion). In our work, we are 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-González et al. 2003). Students have to indicate their intention to draw in the workspace, and when one student is drawing the other students cannot. Learning systems that have a shared workspace also often 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 openers 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 (Vizcaíno et al. 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 and Corbett 2006). In cognitive tutors, students are asked to perform several steps to complete each problem-solving task. These steps can be considered as

Table 1 Tasks and assistance provided in several ACLS systems

System	Task elements	Assessment method	Goals
COLER (Constantino-González et al. 2003)	Modeling, shared & private workspace, chat (classifiers)	Solution structure, individual contributions	IP, RI
COLLECT-UML (Baghaei et al. 2007)	Modeling, shared & private workspace (phases), chat (classifiers)	Solution structure, individual contributions, solution quality	IP, RI, DM, DL
COMET (Suebnukarn and Haddawy 2004)	Medical problem-based learning in shared workspace, chat (unstructured)	Action counts, action sequences, student expertise, solution quality	IP, RI
CycleTalk (Kumar et al. 2007)	Shared workspace (different phases), unstructured chat	Chat counts, keywords in chat, parsing of chat	RI, TO, DL
Group Leader (Israel and Aiken 2007)	Programming with chat (sentence starters)	Count dialogue acts, keywords, sequences of disagreement	RI, DM, RC
HabiPro (Vizcaíno et al. 2000)	Editing computer programs using chat, shared workspace	Solution quality, individual expertise, help type requested, chat counts, keywords	IP, TO, RI, DL
LeCS (Rosatelli and Self 2004)	Case study (phases) in chat (sentence starters), shared text editor, solution representation	Length of time to complete a step, chat counts, solution	RI, DL
MArCo (Tedesco 2003)	Graphical planning in shared workspace, chat (dialogue games)	Logical conflict between student utterances	RC
OXEnTCHÊ (Vieira et al. 2004)	Chat (sentence starters)	Chat counts, keywords	TO, RI

The tutoring goals described are information pooling (IP), reciprocal interaction (RI), dialogue management (DM), task orientation (TO), reaching consensus (RC), and domain learning (DL)

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 students' 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 and Koedinger 2002). 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 function similarly: in addition to informing students about the communication acts expected, adding sentence starters to an interface can make a student's communication intention visible (e.g., Israel and Aiken 2007). Private and shared workspaces can make the discrepancy between an individual's private reasoning and their group contributions visible (e.g., Constantino-González et al. 2003), and thus provide input to an adaptive system.

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 Table 1 for an overview). In many ways, these ACLS systems are very different from each other. 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-González et al. 2003), some is provided on demand (Vizcaíno 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 and Aiken 2007) to decision trees (Constantino-González et al. 2003) to constraints (Baghaei et al. 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 described using a collaboration analysis scheme developed by Meier and colleagues (Meier et al. 2007), where student interaction is rated on 9 dimensions. Some systems attempt to improve student interaction on Meier and colleagues' dimension of *information pooling* (IP), i.e. how much students share their knowledge with their groupmates (Constantino-González 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 support Meier and colleagues' dimension of *dialogue management* (DM), or how students execute conversational acts. Assessment in this area is based on chat actions; sentence classifiers are used to count utterances of particular types or even create a model of 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 and 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-González et al. 2003), chat contributions (e.g., Vieira et al. 2004), or the length of time since students have contributed last (Rosatelli and Self 2004) in order to assess this dimension. However, systems do not generally use all three metrics at once. In addition to providing collaboration feedback on aspects of student interaction, some systems also provide task-related feedback that targets *domain learning* (DL). This feedback is generally provided in a manner similar to individual learning systems. For example, *Cycle-Talk* (Kumar et al. 2007) engages collaborating students in tutorial dialogues that are identical to those used for individual learners.

The different models and feedback in a given ACLS system are not often integrated, pointing toward an opportunity for the advancement of the systems. In particular, different types of collaborative and domain feedback are often kept separate by design, with each type of feedback appearing to students at different times during the collaboration. For example, *GroupLeader* (Israel and Aiken 2007) has three types of feedback: get back on topic, incorporate a single idea per post, or re-evaluate a conflict. In the system, there is never a case where it is appropriate for the two types of feedback to be given at the same time, avoiding the issue of how to decide between multiple feedback options. Although this configuration is a good initial policy, as systems begin to provide more comprehensive support to student collaboration, keeping the feedback separate in this way will not scale. Furthermore, the models and assessment mechanisms underlying the feedback are often kept separate within ACLS systems. *COLER* (Constantino-González et al. 2003), 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. More notably, task-related models are rarely used to augment collaboration models, 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 provide information on 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. One system that does integrate domain and collaboration information is *COMET* (Suebnukarn and Haddawy 2004), where the next participant in a collaborative dialogue is selected based in part on which student has the domain expertise to make a contribution. The effect of this assistance on users has not been explored. One next step in ACLS design

is to provide more complex assistance by integrating different types of models of interaction and different types of feedback. In particular, integrating domain and collaboration models might have large benefits in providing interaction support that is sensitive to the problem-solving context.

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 (McNamara et al. 2007). *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 et al. 2006). 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 and correctness of the student action. Other researchers have explored similar methods of augmenting an intelligent tutoring system with agents that improve student motivation (Del Solato and 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 support 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 that can be flexibly integrated with other components using a standard framework (Krueger 1992). At a minimum, the way a system is divided into components has an impact on reuse, because each 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; Vizcaíno 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., Vieira 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 and 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 and Haddawy 2004). One or more *translator* components are sometimes also 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 and Aiken 2007; Vieira 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 et al. (2007) discuss 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. However, 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 et al. 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 and Self 2004). Although the described designs for reuse can make it easier to increase the 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 components: a "Participation Monitor", a "Difference

Recognizer", and a "Diagram Analyzer" (Constantino-González et al. 2003). This division of tutoring components by functionality can make it easier to incrementally add tutoring complexity by integrating multiple tutoring components, particularly if there is a framework in place so that new tutoring models can be integrated with existing tutoring models.

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 and Koedinger (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 and Koedinger (1996) 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 et al. 2006). Allowing multiple tutors, and providing an integration framework for the tutors, might allow us to provide more complex tutoring to collaborating students.

2.4 Evaluation of adaptive collaborative learning support 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 (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 and Haddawy 2004). 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-related and usability studies rather than controlled experiments. To inform the development of the adaptive component of *LeCS*, data from dyads interacting using the *LeCS* interface were collected and analyzed (Rosatelli and Self 2004), and after *OXenTCHÊ* had been implemented, the usability and the benefits of the assistance were rated by student users (Vieira 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*,

Table 2 Evaluations of ACLS support

System	Evaluation purpose	Evaluation specifics
COLER (Constantino-González et al. 2003)	Feedback validation	Expert ratings of system support, comparison of expert & system support
COLLECT-UML (Baghaei et al. 2007)	Controlled experiment	2 conditions (adaptive collaboration support vs. no collaboration support), classroom study, effects on learning and interactions
COMET (Suebnukarn and Haddawy 2004)	Model validation	Predict individual & group solution paths
CycleTalk (Kumar et al. 2007)	Controlled experiment	2 (collaborative, individual) × 3(adaptive, static, no support) design, classroom study, effects on learning & interactions
GroupLeader (Israel and Aiken 2007; McManus and Aiken 1996)	Model validation, usability study	Assess student dialogue acts, single-condition evaluation of the effects of the system on learning
HabiPro (Vizcaíno et al. 2000)	Model validation	Assess need for assistance, off-topic behaviors, & passivity
LeCS (Rosatelli and Self 2004)	Design-related study	Students use a non-adaptive system to inform design
MArCO (Tedesco 2003)	Usability study	Students use adaptive and non-adaptive versions of the system to explore its effects
OXEnTCHÊ (Vieira et al. 2004)	Usability study	Usability, student ratings of system assistance

Evaluations range from technical validations of the models behind the systems, usability studies of interactions within the system, and controlled experiments to evaluate student learning

Baghaei et al. (2007) compared an adaptive collaboration support condition to a no 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 et al. (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 significantly higher 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 their potential learning benefits.

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. Large amounts of data are often required to develop the assessment components of the systems, but the data can be difficult to collect. After expending 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 (Carnegie Learning 2009). Tutor data is often mined in service of investigating learning science hypotheses and ultimately informing the improvement of intelligent tutoring systems (Beck et al. 2004). Similarly, it has become common practice to perform embedded experiments, making small modifications to already deployed tutoring systems (Mostow and 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. Taking advantage of these relationships is a main goal of the Pittsburgh Science of Learning Center, which connects researchers and classrooms, and instruments classrooms so that it is easier to collect data and evaluate learning interventions (PSLC 2009). 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 of adaptive support.

2.5 Outlook

Our goal is to build a framework 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 generally not 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. Further, facilitating this integration would make it easier to combine domain-specific task models with domain-general models of good collaboration, enabling context-sensitive collaborative tutoring across multiple tasks. 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 a framework for ACLS that facilitates controlled research of different types of adaptive support, and for this purpose we introduce the Collaborative Tutoring Research Lab (*CTRL*). CTRL 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 and Gaßner 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* (Berge and Slotta 2007), 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.

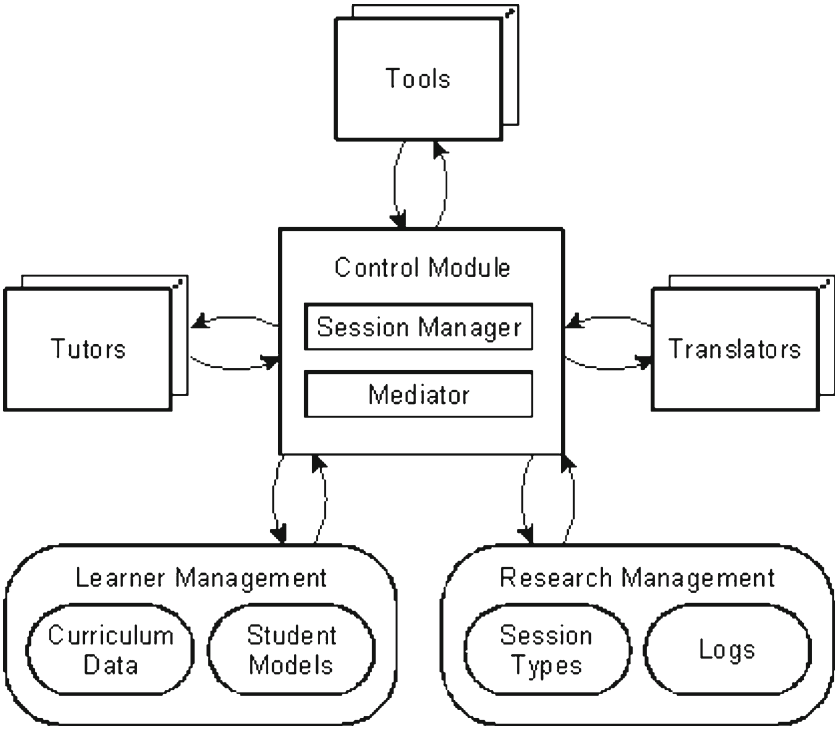

Fig. 2 High level overview of CTRL. CTRL consists of tool, tutor, and translator agents, learner and research management data stores, and a central control module

3 The Collaborative Tutoring Research Lab (CTRL)

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 framework, 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 a teammate would be able to use results provided by a domain tutor about whether the facts shared were correct. CTRL 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, the way they interact with each other, and the way they can be integrated. The actual design, implementation, and evaluation of a peer tutoring scenario using CTRL is described in Sect. 4.

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

1. *Tools*: Used by the student to take problem-solving actions
2. *Tutors*: Provide students with 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 (session types)
6. *Control Module:* Constructs and manages collaborative sessions, both on a problem-to-problem level (session manager) and on an action-to-action level (mediator)

The focus of CTRL is on facilitating interactions between tool, tutor, and translator components, and we define and discuss each of those components in more detail in Sect. 3.1. In Sect. 3.2, we describe how the various components communicate with each other. Sect. 3.3 outlines how the control module interacts with the research management store 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 chemistry experiments. CTRL allows any number of tools to be involved in the learning scenario. Multiple users can collaborate remotely while each one uses different tool components. 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. Tool components contain the user interface, a domain model, and meta-knowledge of tutoring. The interface is the point of interaction between the user and the system. The domain model is present so that the tool can update its state without input from an additional component. 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. For example, in a chemistry simulation environment, the interface might allow students to mix different chemicals, and the domain model might calculate and display the result of mixing the chemicals. Although tools should be able to share domain models, this behavior is currently not explicitly supported by CTRL, in part because of our focus on using pre-existing components that already have a domain model. Tools also contain meta-knowledge so that they can convert feedback from a tutor agent into a format appropriate for display. Thus, tutors can be used with any tool because they do not need to send tool-specific messages. When the chemistry simulation tool mentioned above receives a hint message from the tutor, it might display it in a pop-up dialogue in the interface, while if a collaborative discussion tool receives

the same message, it might display it as part of the chat interaction. The functionality that we have described is ideal, but it is likely that many pre-developed tools we may want to use will not incorporate all functionality, and may be 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. For example, if a particular collaborative discussion tool does not know how to handle a hint message, a translator would need to be built to convert the abstract message (e.g., giveHint[hint]) into a format the tool understands (e.g., displayInChat[hint]). This aspect of translator functionality is very much in line with the translators discussed in Ritter and Koedinger (1996) 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. For example, a translator could be used to allow some actions made by one student to appear on the other student's screen, but not others. 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. 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.

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 skills based on the model. 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. Tutor components should contain an expert model, a feedback model, and a student model. Like in regular intelligent tutoring system functionality (as described in VanLehn 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 preexisting 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, a student with an id of jmiller entering 25 in a table might be represented as (student = jmiller, selection = cell A1, action = enterValue, input = 25). 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 the student action in cell A1 (indicating it was correct), send a feedback message for encouragement (e.g., "Keep it up! What goes in cell A2?"), and increase the value of the relevant skill (e.g., set the skill "entering values in a table" to 60%). 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, which is a conventional data structure containing any number of attribute-value pairs.

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 CTRL, here we present an in-depth example of how those messages might be used by the different components (see Fig. 3). The example includes two tools (representing two collaborating students, Bo and Jan), two tutors (representing a domain and collaborative tutor), one translator to implement the shared collaborative workspace, one research management component, and the mediator subsection of the control model. In the

Table 3 Messages passed 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

Messages are used to for session management tasks like moving to the next problem, but also for tutor-tool interactions within the problem

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 control module. Meanwhile, the domain (math) tutor evaluates the user action, and sends its feedback to the control module, 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 control module using a *processFeedback* message. The control module has now received messages from the translator, the collaboration tutor and the domain tutor. The control module integrates the messages, passes the *scriptInteraction* message along to both tools, and then sends the feedback message to Jan's tool, as specified by the integration logic in the control module. 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.

We have explicitly chosen to leave some elements necessary for implementing a computer-supported collaborative learning system unspecified, because they are outside of our main focus. As the system is distributed, some components of the

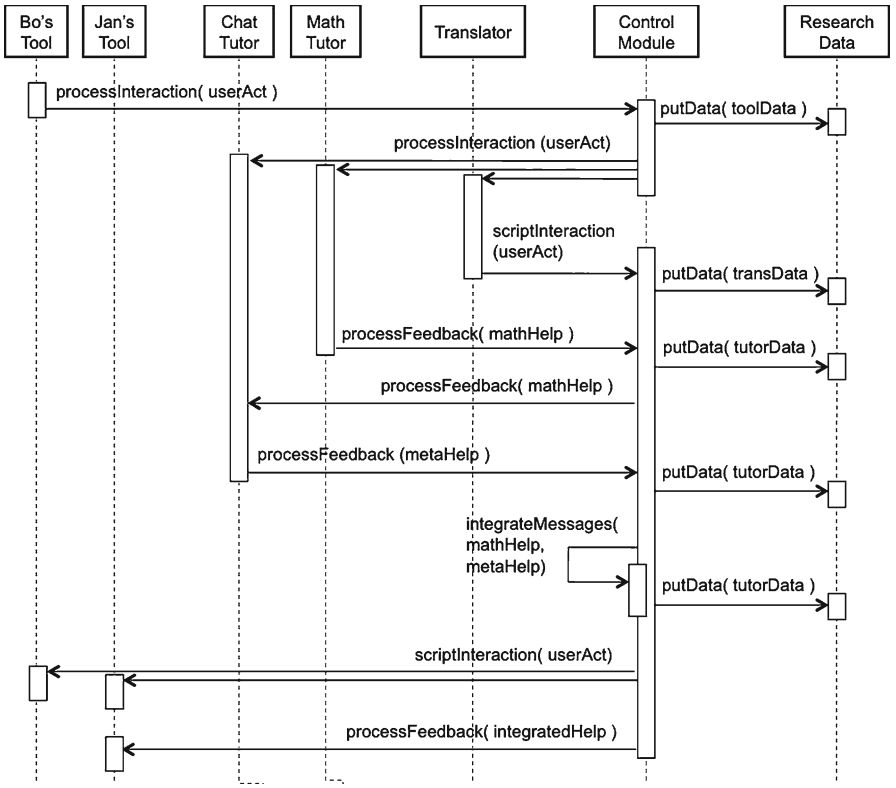

Fig. 3 Message-passing between two tools, two tutors, and a translator. Each tool represents a collaborating student. One tutor supports student interaction and one supports problem-solving

system (e.g. the control module) will run on a central server, and some components (e.g. the tool components) will run 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 Sect. 4.

3.3 Component integration

In addition to illustrating how messages are passed between components, there are several notable elements of the above example that highlight the centrality of the control module during a session. All messages sent go through the control module, 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 control module is in control of which components are involved, where messages get sent, and how messages are integrated. Using 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, facilitating the integration of different tutor components. While CTRL is not the only architectural framework to use a federated system (see Rosatelli and Self 2004; Mühlenbrock et al. 1998), its contribution is that it focuses specifically on integrating different tutor components and on the efficient implementation of comparison conditions.

The 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 CTRL 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 CTRL in Sect. 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 CTRL, each semantically meaningful action occurring within a component is sent to the control module, which transforms the action into xml, and sends it to a data store in the research 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 protocol (PSLC 2009), which records semantic-level messages sent from tool and tutor components. These tool and tutor logs follow the concept of a transaction described by VanLehn et al. (2007), where a user action and the tutor response to the action are linked. In our framework, a *processInteraction* 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 Fig. 4). 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 represented, as the tutor message contains the ID of the tool message that triggered it. Logs 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 CTRL 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. (2007). Thus, an additional type of message is supported: a *scripting* message, logged whenever a module changes the problem state of a tool. In this case, the student interaction parameters, the timestamp, the relevant context message id, and the relevant tool message id are logged. Second, because CTRL 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 of the user, and the role of the user within that session. We incorporate this information into the context message, which logs the learning environment settings. Third, because CTRL supports multiple tool responses, the relevant metaphor for analyzing the data is not a single tool-tutor transaction but a chorus of responses to a tool action. Not only does each tutor response need to be logged, but also the final message constructed by the mediator to be sent to each tool.

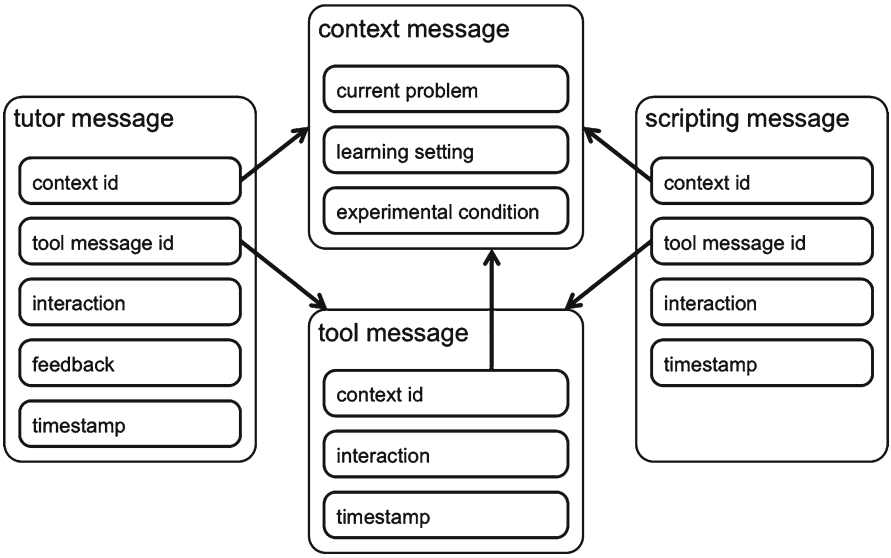

Fig. 4 Logging format for student-tutor interaction. Logs consist of context messages, tutor messages, tool messages, and scripting messages

3.4 Outlook

Ideally, CTRL captures rich process data, integrates feedback from multiple tutor components, and makes 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 pre-existing and custom-built intelligent tutors can be incorporated into the system 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. Finally, because 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 CTRL that demonstrates these features.

4 Instantiation of CTRL

We demonstrated the suitability of CTRL as a research platform by using it as the foundation for conducting a controlled study 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 called APTA (Adaptive Peer Tutoring Assistant) 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

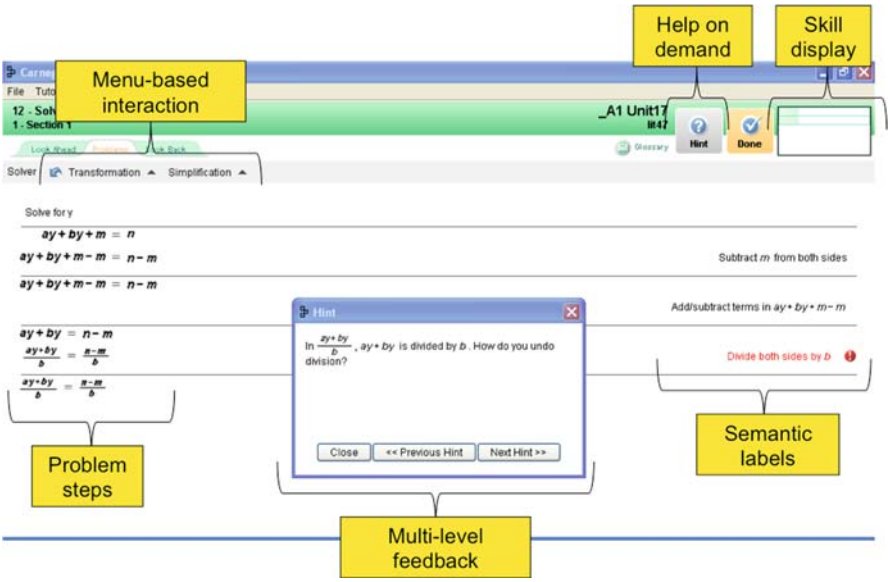

Fig. 5 Individual version of the CTA. Students solve problems in the equation solver, and receive hints and feedback from the cognitive tutor. © 2009 Carnegie Learning. Used with permission

conditions, using an instantiation of CTRL. Finally, we describe a controlled classroom study in which we evaluated the adaptive peer tutoring condition. 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.

4.1 Intervention design: peer tutoring in the context of the Cognitive Tutor Algebra

APTA is designed as an addition to the Cognitive Tutor Algebra (CTA). Figure 5 shows the literal equation solving unit of the CTA. Students use menus in an equation solver tool to manipulate the equation, selecting operations like "add x" or "combine like terms". The semantic label for the operation then appears on the right side of the screen. For certain problems, students have to type the result of the operation in addition to selecting it. 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 visual feedback in the interface, and often a message describing their misconception. At any point, students can request a hint on the next step of the problem. The CTA monitors student skills, reflects them in a skill display, or *skillometer*, and selects problems based on student skill mastery. As students may acquire shallow conceptual knowledge while using tutoring systems, recent efforts have augmented cognitive tutors with activities that encourage elaboration. There are promising early results on adding supported collaborative activities to the CTA (Diziol et al. 2008).

We augmented the CTA with a reciprocal tutoring script. When students act as peer tutors they benefit because they are reflecting on the current state of their knowledge

and using it to construct new knowledge (Roscoe and Chi 2007b). As these positive effects are present even if peer tutors have low domain knowledge, researchers 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 in long-term interventions integrated into classroom practice (Fantuzzo et al. 1989). Unfortunately, such gains are not always seen, possibly because students do not often exhibit positive peer tutoring behaviors spontaneously (Roscoe and Chi 2007a). 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, training students to deliver conceptual mathematical explanations and elaborated help had a significantly positive effect on tutor learning (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. If not, there may be cognitive consequences (tutees cannot correctly solve problems; Walker et al. 2007) and affective consequences (when students feel that they are poor tutors they become discouraged; Medway and Baron 1997).

In APTA, 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 anticipation of tutoring may lead students to feel more accountable for their knowledge and therefore attend more to the domain content during preparation. Pair members are each 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. Sara would solve the second problem just as though she was using the individual cognitive tutor, by manipulating the menus in the Equation Solver 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 and 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 own 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 instant messaging tool, and we expected that providing this functionality would facilitate elaborated discussion between the students. Furthermore, Bob has access to the problem solution in an interface tab, in order to provide him with fixed domain assistance during tutoring (see Fig. 6).

We used the CTA models to further provide adaptive collaboration assistance to the peer tutor, using a *meta-tutor*. In a pilot study with unsupported students using

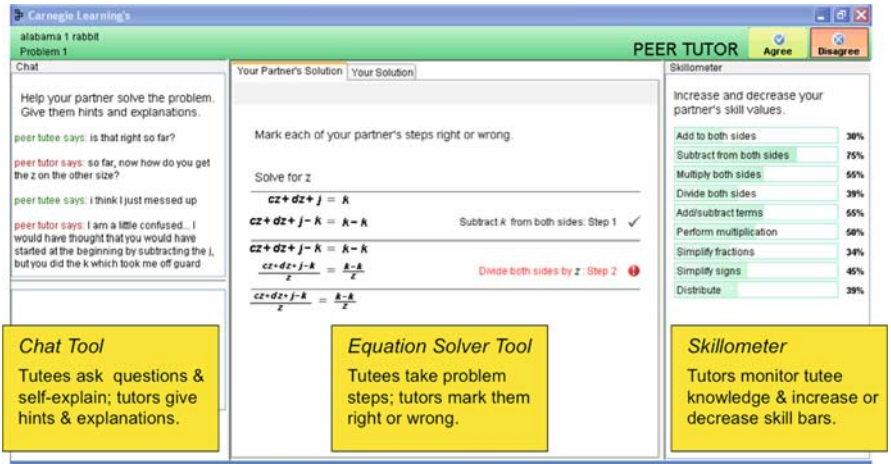

Fig. 6 Peer tutor's interface. As the tutee solves problems in the equation solver window, the peer tutor can mark steps right or wrong. The peer tutor can also give tutees feedback by increasing and degreasing their skill bars. Students can talk in the chat window

the peer tutoring script, we found that peer tutors had difficulty giving correct help to their tutees, and tutees solved few problems correctly (Walker et al. 2007). Therefore, we focused our first attempt at adaptive assistance for the peer tutor's corrections of the peer tutee's 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 (indicating that the tutee believes that the problem has been solved)
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 or she 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 that the peer tutor marks a step right when it is in fact correct, and marks a step wrong when it is incorrect. Path 3 is 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 reflect on their misconceptions and then deeply process the feedback as they attempt to communicate it to the tutee. In order to support Path 3, we also 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 Fig. 7). The goal of providing the hints and feedback is not simply to force the peer tutor to reproduce all help the CTA would have provided. The meta-tutor

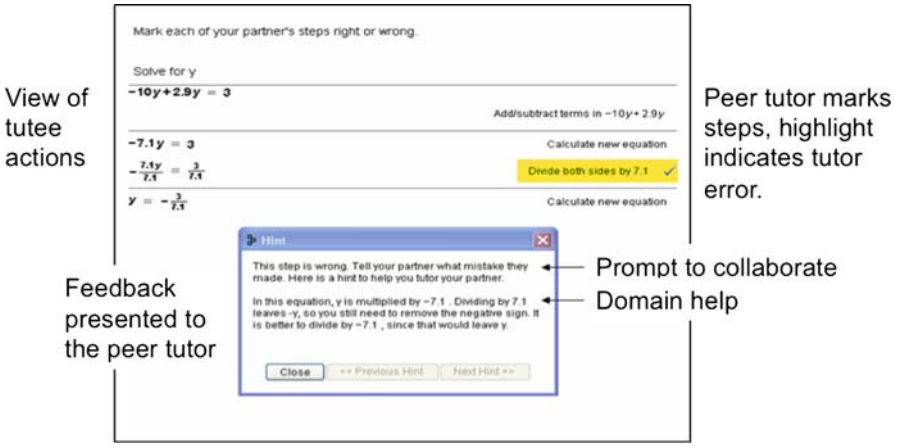

Fig. 7 Feedback delivered to the peer tutor. Once the peer tutor marks a step incorrectly, the step is highlighted in the interface. The peer tutor then receives feedback containing a prompt to collaborate and domain help originally designed for individual learners

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.

Our educational psychology research goal was to evaluate the effects of the adaptive assistance on student collaboration in APTA by comparing it to two comparison conditions. 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 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 Sect. 3. Then we present the empirical study and its results.

4.2 Implementation of study conditions with CTRL

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 the CTRL framework, with a mixture of custom-implemented components and components that were originally part of the CTA. The *adaptive peer tutoring condition* included two tool components (the peer tutor's interface and the peer tutee's interface), a translator component (to echo actions from one tool to the other tool), and two tutor components (a 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). The *fixed peer tutoring condition* included the same two tools as the adaptive condition, and the translator component. The *individual use condition* included the original CTA tool and tutor components, using a tool similar to the peer tutee's tool and the domain tutor. All conditions included a learner management component, a research management component, and a control module to integrate all the components. Conditions were implemented in Java.

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 and Koedinger's (1996) clean separation between tools and tutors, development constraints led its current state to evolve from this ideal. Therefore, our first step to being able to use the CTA tool components was refactoring them so that the tool functionality was separate from the tutor functionality. Because this process entailed 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 peer tutor's and peer tutee's interfaces.

The *translator component* was a custom-made component designed to facilitate collaboration between two users. This component functions by receiving all *processInteraction* messages and converting them into corresponding *scriptInteraction* messages before sending them back to the tool components through the control module. 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. In the CTA, the tool needs permission from a tutor to effect certain actions (e.g., to create a point on a graph). 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 CTA tools, and is therefore not a general component for facilitating collaboration (which, given the goal of working with existing 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 (e.g., a solver action) and peer tutor response (e.g., the peer tutor marked it incorrectly *wrong*) in choosing from a fixed set of collaboration-oriented meta-feedback (e.g.,

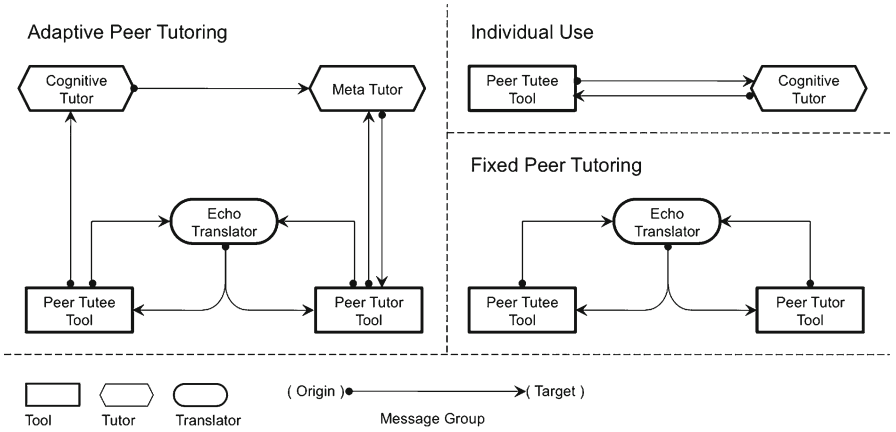

Fig. 8 Message passing logic for adaptive peer tutoring, fixed peer tutoring, and individual use conditions

"Your partner is actually right. Why don't you talk to them about why they took this step"). Then, if the domain tutor has appropriate feedback, the meta-tutor appends the cognitive tutor message to the meta-feedback message. The tutoring model sends a *processFeedback* 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 the peer tutor must respond 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 work in a similar manner, combining the cognitive tutor hint on the step with a prompt to collaborate. The meta-tutor is domain-independent, and thus could be effective in combination with any intelligent tutor, as long as a translator exists to translate the intelligent tutor messages into an appropriate message format.

In general, components communicate using the CTRL message protocol, and the way components interact in a given session is defined in the control module. All peer tutee solver actions, peer tutor correction actions, peer tutor skill ranking actions, and student chat actions are logged as tool messages by the control module. All cognitive and meta tutor feedback and hints are logged as tutor messages. See the left hand side of Fig. 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 a bug rule fires, 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. The right hand side of Fig. 8 shows the message passing logic for the other two conditions: fixed peer tutoring and individual use.

As the logic of which components are involved in the session and how they communicate exists in the control module, it is simple to use the module to implement

the relevant conditions. The components involved were defined in the same manner as in the CTRL framework, where all the components involved in a session and their component types are enumerated. However, instead of the message passing logic being defined in a rule-based manner, it was initially defined in the form of several *message groups*, each comprising an originating component, a target component, and a priority (represented pictorially in Fig. 8). Message groups can be considered 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 transition to a rule-based format in the future. Three session types were created that corresponded to the three conditions, so it was simple to switch from one condition to another.

In CTRL, 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 the 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 Empirical 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. A description of the study can be found in Walker etal. (2008). Participants were 62 high-school students from five second-year algebra classes, taught by the same teacher. The high-school used the individual version of the CTA as part of regular classroom practice. Students in the collaborative conditions were assigned to pairs by the classroom teacher, who was advised to group students of similar abilities who would work well together. Students from each class were randomly assigned to one

Table 4 Student pretest scores, delayed test scores, and gain score

	Pretest score		Delayed test score		Pre-delayed gain	
	M	SD	M	SD	M	SD
Adaptive peer tutoring	0.82	1.08	2.82	1.78	0.29	0.19
Fixed peer tutoring	0.90	0.88	3.60	2.17	0.30	0.51
Individual learning	1.28	1.60	3.67	1.78	0.26	0.50

of the three conditions. The total number of participants included in the analysis was 39 (11 in the adaptive peer tutoring condition, 10 fixed peer tutoring condition, and 18 in the individual use condition). There were an odd number of students in the adaptive condition because we retained students in the analysis who had an absent partner during an intervention day but were placed with a new partner in the same condition. The study took place over three weeks. Early in the first week, students were given a pretest. The intervention then occurred over two 70 minute class periods, each a week apart. On both intervention days, students in the peer tutoring conditions spent the first half of the period in the preparation phase, and the second half taking turns tutoring each other in the collaboration phase. Students switched roles between tutor and tutee after every problem. Students in the individual use condition simply used the CTA as during regular classroom practice. The week after the intervention, students were given a posttest. Two weeks after the posttest, students were given a delayed posttest to assess their long-term retention. We logged all tutor actions, tutee actions, and intelligent tutor responses. The log data allowed us to extract process variables such as incorrect attempts made by students, help accessed by the peer tutor, help communicated by the peer tutor, and problems completed.

In the remainder of this section, we look at how the implementation of the three experimental conditions, facilitated by CTRL, helped us to gain insight into the learning effects of adaptive support for peer tutoring. First, we describe how multiple streams of interaction data, in connection with outcome measures, provided us with insight into the peer tutoring process. Next, we examine how the integration of the domain and 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. Because this analysis is exploratory and intended as a demonstration of the goals of CTRL, our discussion includes trends in addition to significant results. For reference, Table 4 provides the pretest and delayed test scores of the three conditions. Gain scores were computed using the following formula: (delayed score − pretest score) / (total points possible − pretest score). For negative gain scores, the formula we used was: (delayed score − pretest score) / (pretest score).

4.3.1 Benefits of collecting rich log data

To demonstrate the benefits of collecting rich log data, we focus on one particular result in the adaptive peer tutoring condition: relating student impasses to the learning gains between pretest and delayed test. 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. This hypothesis can be evaluated through the integrated stream of data collected using the CTRL logging protocols. In the adaptive condition, the number of incorrect problem solving attempts *per problem* on the part of the tutee were negatively correlated with tutee learning ($r(9) = -0.561$, $P = 0.072$), as were the percentage of incorrect attempts to move to the next problem out of total attempts ($r(9) = -0.667$, $P = 0.025$). 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. However, surprisingly, tutor learning was *positively* correlated with tutee incorrect problem solving attempts ($r(9) = 0.523$, $P = 0.099$) and tutee percent incorrect done attempts ($r(9) = 0.652$, $P < 0.030$). These results, while correlational, suggest that peer tutors benefit from actively processing tutee errors, which was similarly demonstrated by studies on learning from erroneous worked examples (Große and Renkl 2007). Reaching this insight required integrating problem-solving data, information about student correctness (using cognitive tutor models), and outcome data.

The relationship between the chat logs, problem-solving logs, and correctness information also provides us with insight that would not have been available had we only had one source of interaction data. Table 5 displays a student interaction immediately after the peer tutee has taken an incorrect done action and the peer tutor has incorrectly agreed. The equation the students are working on is "$t = f/(1 - .75)$", with the goal of solving 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 5 took 10 min. If we were to look only at the left hand column of Table 5, 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 answers 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 0.33, suggesting that some of this active processing was beneficial, but also that there was more room for improvement.

4.3.2 Benefits of integrating domain and collaboration support

The addition of adaptive feedback that incorporates both domain support and collaboration support (via the meta-tutor) can provide us with insight into how providing

Table 5 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)

adaptive feedback affects the peer tutor's behavior and tutee learning, especially in comparison to domain support implemented in a fixed manner. In this section, we use the rich log and outcome data to make a direct comparison between student use of fixed assistance and adaptive assistance, and as a result, gain insight into the relative merits of each assistance type in this context. This comparison would not have been possible had we not been able to leverage the existing CTA models in order to implement the adaptive domain support.

In the 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 looked only at the 9 students who chose to use all forms of assistance when in the role of the peer tutor. As evident from Table 6, students accessed more fixed assistance than adaptive assistance. However, there were differences in the relationship between the two types of assistance received by the tutor on the gains of the student in the peer tutee role. Communicating adaptive assistance was positively correlated with tutee learning ($r(7) = 0.786$, $P = 0.115$), while failing to communicate adaptive assistance was negatively correlated with tutee learning ($r(7) = -0.803$,

Table 6 Amounts of adaptive and fixed assistance communicated and not communicated

	Adaptive assistance		Fixed assistance	
	M	SD	M	SD
Assistance communicated	1.44	1.81	3.56	3.84
Assistance not communicated	1.44	2.01	5.67	6.22

$P = 0.102$). Surprisingly, communicating domain feedback to the tutee after accessing fixed support (i.e. checking problem answers) was negatively correlated with tutee learning ($r(7) = -0.925$, $P = 0.024$).

We can retrieve from our log data examples from each of the three different relevant assistance cases (adaptive assistance communicated, adaptive assistance not communicated, and fixed assistance communicated) to illustrate what may be occurring. The following is an example of the peer tutor receiving feedback on a step, and not communicating it. After marking a step right, the peer tutor received 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 said: "that doesn't look right, im sorry I suck at math lol", and then "k, nevermind." The peer tutor did not respond. Next, the peer tutee clicked done, the peer tutor agreed, and the peer tutor was given another feedback message saying the problem is not done. This message was also not communicated to the tutee. Thus, not only were tutees not getting the assistance needed, but they were getting misleading feedback. To the tutee, it appeared as if the steps were correct, even if they were not. This example can be compared to the following example where the feedback received was communicated:

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 together cleared up a misunderstanding about which aspect was incorrect. In the final example, fixed assistance was communicated to the tutee: After the tutee took an incorrect step dividing both sides by $q+r$, the peer tutor checked the answers and said, "divide both sides by $q + s$." The tutee then promptly undid his last step and performed 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 or her error.

4.3.3 Use of comparison conditions

Because CTRL 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 and 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 learning gains in the adaptive condition appear satisfactorily high, with a mean gain score of 0.29 ($SD = 0.19$) between the pretest and the delayed posttest. However, comparing the learning improvement across all three conditions, we see that the adaptive condition score is not different from the fixed condition score ($M = 0.30$, $SD = 0.51$) or the individual use condition ($M = 0.26$, $SD = 0.50$). In fact, an ANOVA reveals that the gain scores are not significantly different ($F(2, 36) = 0.033$, $P = 0.967$). Then, even though the learning gains across the three conditions are similar, 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 ($M = 47.0$, $SD = 30.2$) was 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 ($M = 17.7$, $SD = 6.69$). A logical hypothesis may be that students in the individual condition learned by solving many problems quickly but shallowly, whereas in the collaborative conditions, students learned by solving fewer problems slowly but deeply. In general, 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 CTRL, 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, we 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 CTRL facilitated the learning sciences research that we conducted.

5 Implications of CTRL

We could have much more sophisticated systems to provide adaptive collaborative learning support if we had an architecture for plugging together the many excellent

and complementary systems that already exist. The main claim made by CTRL is that several different tool and tutor models can be combined to form a variety of different collaborative scenarios, and then components can be systematically removed to create experimental control conditions. We have already presented the Adaptive Peer Tutoring Assistant (APTA), one extended example of a collaboration activity that can be supported under CTRL. In this section, we present two other hypothetical examples in order to further illustrate the potential use of CTRL. The first example also involves a collaborative extension of the CTA, but in a different setting: here, students engage in collaborative problem-solving rather than peer tutoring. We demonstrate that even if we were to use the same core code elements that we used in the peer tutoring extension, we could create a remarkably different collaborative experience for students. In the second example, we take a different approach, demonstrating how an existing collaboration script could be used as the jumping off point for adaptive collaborative learning support, rather than an existing intelligent tutoring system. We describe how the script could be implemented using CTRL and how tutors from other systems might augment the script. Then, we map out the space of control conditions made possible by the augmented script. To conclude, we discuss the limitations of CTRL as a framework.

5.1 Implementing the collaborative problem-solving script in CTRL

In our first example we use a collaboration script developed by Diziol et al. (2008), called the collaborative problem-solving script. In this script, two students are put in pairs to work together on a "systems of equations" problem on a single computer. While collaborating, students receive two types of feedback: domain-specific task-related feedback on their problem-solving, and domain-general collaborative feedback encouraging them to engage in effective learning strategies. This script was shown to improve deep conceptual learning, but was implemented in a face-to-face computer-supported setting rather than using computer-mediated collaboration, limiting the ability of the provided adaptive support to assess and provide feedback on the collaboration.

This script could be implemented within CTRL by primarily using existing CTA components. Each collaborating student would need a tool component, which consists of a graphing widget (already present in the CTA), a spreadsheet widget (already present in the CTA), and an instant messaging client (already incorporated in APTA). The tool components would broadcast *processInteraction* messages using selection-action-input triples (as described above in Table 3). The collaborative problem-solving script could also use APTA's translator to allow graphing and worksheet actions made by one student to appear on the other student's screen, by sending *scriptInteraction* messages. Finally, there are also two tutor components involved in this scenario: a cognitive tutor that provides task-related support, and a collaborative tutor that provides support for the student interaction. Like in APTA, the cognitive tutor would be taken from the existing CTA; unlike APTA, this tutor communicates directly with the students, by broadcasting problem-solving feedback to both students' screens using a *processFeedback* message. The collaborative tutor would be a simple custom-built

component that uses input from the CTA models to detect ineffective learning strategies on the part of the student, and respond appropriately. For example, if students make several errors in a small amount of time, the tutor would classify their strategy as a "trial-and-error" strategy, and send feedback using a *processFeedback* message. All the components could be combined to form the collaborative scenario by adding a new session type in the mediator, which specifies that each tool component sends messages to all tutor and translator components, each translator component sends messages to all the tool components, and each tutor component sends messages to all tool components.

Reimplementing the collaborative problem-solving script within CTRL would have several advantages for the expansion and evaluation of the script. First, the collaborative version of the script could be enhanced using components already developed for the individual version of the CTA. To illustrate we use the example of the Help Tutor: an addition to the CTA developed by Aleven et al. (2006). The Help Tutor accepts student process data and knowledge assessments as input (e.g., time between steps, hint requests, probability that students know a skill). It then provides as output a classification of their help-seeking behavior, and direct feedback related to negative help-seeking behavior. The Help Tutor is a natural fit with the collaborative problem-solving script, in that it is a more sophisticated and potentially more effective way of diagnosing a variety of hint abuse and trial and error strategies that could then be used as input to the collaborative tutor module. The collaborative tutor module could then respond to these individual problem-solving errors with feedback prompting the students to collaborate more effectively in order to overcome their impasses. Improving the collaboration script in this manner would simply require the integration of the Help Tutor within the CTRL framework; the tutor would need to receive *processInteraction* messages, send *processFeedback* messages, and be added to a session type in the mediator. Second, reimplementing the collaborative problem-solving script within CTRL makes it easier to generate experimental control conditions to investigate educational psychology research questions. For example, by adding different session types to the mediator, the tool component of one partner could be removed to create an individual learning scenario, feedback from the collaborative tutor could be removed such that only cognitive assistance is provided, and feedback from the cognitive tutor could be removed such that only collaborative assistance is provided. Using different combinations of components, researchers can investigate questions such as: What are the effects of cognitive and collaborative assistance to collaboration compared to only cognitive assistance to collaboration? What are the effects of adaptive assistance to collaboration compared to adaptive assistance to individual learning? As a whole, APTA and the collaborative problem-solving script overlap regarding the CTA tools and domain models used, but create very different collaborative scenarios.

5.2 Augmenting the learning protocol approach in CTRL

Our next example is intended to show the versatility of CTRL, and uses an existing collaboration script as its basis rather than an existing intelligent tutoring system. We base the example on a simple computer-supported collaboration script outlined by

Pfister and Mühlpfordt (2002), called the *learning protocol approach*. In this approach, three to five students attempt to comprehend and remember the content of a section of text by discussing it in a chat window. In order to support the cognitive aspects of reading comprehension, students are required to classify their contributions and indicate the previous utterance to which their contribution refers, prior to making a contribution. Further, to support student coordination, the system enforces a strict turn-taking structure to the conversation, selecting which student should speak next based on a predefined order. The learning protocol approach is limited in the ways that many other collaboration scripts are limited, both in that it is not guaranteed that improved collaboration will result from use of the script and in that aspects of the script may in fact overstructure collaboration. There is no guarantee that students will take the referencing or classification activities seriously, and even if they do, they may not generate contributions of sufficient quality to trigger the desired cognitive elaborative processes in themselves or others. Further, forcing students to take turns speaking so rigidly may improve the participation of all students, but it may also decrease student motivation: some students may not be able to participate when they have something they to contribute, and others may become discouraged to be forced to comment when they do not know what to say. Thus, adding adaptive support elements to the script using the CTRL framework has the potential to greatly improve the learning activity.

In order to add components to the learning protocol script within CTRL, the original version of the script would have to be refactored into several identical *tool* components (one for each collaborating student) and two *translator* components. Each tool component would be composed of the text to be analyzed and a chat window, including interface scaffolding (e.g., sentence classifiers and a widget to indicate the reference of the contribution). Tools would broadcast *processInteraction* messages for each user action. One translator would promote cognitive elaboration by sending a *processFeedback* message whenever students do not classify their utterance and mark the reference of their utterance before submitting it. A second translator component would handle the coordination elements of the script, disabling the chat interfaces of the collaborators who do not have the turn to speak, and transferring the turn from one person to another using *setProperty* messages. Once these components have been developed, it is simple to combine them under our conceptual framework by defining a message group between each tool and each translator (e.g., establish two rules: IF a message m was sent by any tool THEN send m to every translator, AND IF message m was sent by any translator THEN send m to every tool).

One key area where adaptive support could augment the script is in evaluating, in real time, whether student contributions to the discussion are of a good quality; essentially, are the script scaffolds having the desired effect? Here, we can turn to an individual intelligent tutoring system for reading comprehension called iSTART (McNamara et al. 2007) to help adaptively support students while using the collaborative reading comprehension script. As part of iSTART, students generate self-explanations based on a particular segment of text, and then the system compares their self-explanation to the segment and surrounding text to assess its quality. The iSTART tutor model could also be used to assess student collaborative utterances, by comparing the student contribution to the reference in the discussion that the students themselves have made. If the contribution is determined to be insufficiently relevant, the student could

be given feedback, to help them improve their utterance. If iSTART can be refactored so that the tutor component is isolated, and then the tutor component can be extended to accept *userInteraction* messages and then to send *processFeedback* messages, it could be integrated with the learning protocol components using CTRL.

Second, adding adaptive components could help mitigate the aspects of the script that overstructure student interaction. For example, rather than forcing students to speak in turns, the system could meet the same interaction goals by making sure that everyone participates relatively equally. To assist this aspect of interaction, we can turn to another ACLS system, LeCS (Rosatelli and Self 2004). LeCS has a model for enforcing participation where if a student is silent for greater than 50% of the time estimated to complete discussing the section, the tutor sends a randomly chosen feedback message encouraging the student to participate. If this model accepted *processInteraction* messages and broadcasted *processFeedback* messages, it could be plugged into CTRL by adding a message group between each tool and this tutor. More interestingly, instead of sending feedback directly to collaborating students, both the participation tutor and relevance tutor could be used as input to a custom-built tutor that tracks the relevance of student statements to particular concepts over time, and then, whenever possible, prompts students who likely have something to contribute to speak next. This concept is similar to the adaptive feedback found in COMET (Suebnukarn and Haddawy 2004). However, because of the framework provided by CTRL, it is easy to make this addition to the system after the custom-built tutor has been constructed, for example by changing the logic in the mediator that sends messages from the participation tutor to the students, and instead send those messages to the custom-built tutor.

Further, once this extended adaptive scenario has been implemented, relevant control conditions could then be generated using CTRL in order to investigate a variety of research questions surrounding the effects of feedback on collaboration. Simply by changing the logic in the mediator for components included in a session type and messages passed between components, the researcher could vary the *type* of support provided (cognitive elaboration support, coordination support, or both), and the *adaptivity* of support for each type (fixed or adaptive). The researcher could also examine more specific questions, such as whether using domain information to augment support for social coordination is better than providing social coordination support alone. These comparison conditions would contribute to a systematic investigation of the current research questions surrounding adaptive collaboration support.

5.3 The scope of CTRL

These two additional examples should contribute to an overall sense of the scope of the CTRL framework and the types of activities for which it is most appropriate. Earlier in the paper, we stated that CTRL focuses directly on the interaction between collaborating students and intelligent support. It is appropriate for use in scenarios where a small number of students have been placed in a group and are collaborating on a particular task. CTRL is not designed to adaptively assign students to particular groups or tasks; that is, it is not a tool for manipulating the preconditions of the interaction. However,

CTRL can be applied in conjunction with a wide variety of different sets of preconditions, once they have been specified, and there is nothing inherent in CTRL that restricts the domains for which it is used. CTRL is also not specifically designed for macro-scripting the interaction (e.g., by specifying a sequence of phases for students to follow, such as alternation between individual and collaboration phases). Although it is possible to implement a macro-script using a translator, the challenges of managing a macro-script are not addressed by the design of CTRL, and there may be simpler ways for doing so within a given adaptive system. Despite this limitation, CTRL can be applied to manage interactions *within* the phases of macro-scripts. Finally, although CTRL could be applied to asynchronous interaction, it was designed with synchronous interaction in mind, and it is likely there are other frameworks more appropriate for managing asynchronous communication. Within these parameters (adaptive or fixed micro-scripting of synchronous interaction between a small number of students given a particular task), CTRL actively facilitates the implementation of different types of interactions.

In determining what is necessary for other researchers to use CTRL, it is important to make the distinction between the conceptual framework itself and our specific implementation using existing CTA components. The mediator component of CTRL is simple to implement, and one could imagine other researchers adopting this concept in order to develop their systems (and we would encourage this!). However, the difficult part of applying CTRL is refactoring the components of existing systems to separate the tool, translator, and tutor functionality; for example, integrating iSTART with the learning protocol approach would depend on what would be required to isolate the tutor component of iSTART. For large and complicated systems whose code has been developed iteratively and by multiple people, this process can be a challenge. Ideally, once the code has been refactored, it would not be necessary to make modifications to the existing components. However, practically, this is not the case; it still can be difficult to interpret and modify the code relating to the existing tutor components, as became clear when a colleague of ours encountered difficulties in his attempt to integrate our code with a simulated student tool. In cases where existing components are used, we need to do more work towards reducing the need for them to be modified. However, as more systems are developed with a component-based approach, CTRL will become more and more effective.

6 Conclusions

We have outlined a conceptual framework called CTRL that supports educational technologists in developing adaptive support for collaboration and educational psychologists in investigating its effects. The framework enables researchers to integrate different types of adaptive support and, in particular, allows domain-specific models to be used as input to domain-general components in order to create more complex tutoring functionality. Additionally, the framework helps researchers to implement comparison conditions by making it easier to vary single aspects of the adaptive intervention through removing tool or tutor components from a system. We demonstrated the use of CTRL by first designing adaptive support for a peer tutoring script, then instantiating

the framework using the adaptive scenario, a fixed support scenario, 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 an experimental study, and the results illuminated the relationship between tutee impasses and tutor learning, the different effects of communicating adaptive and fixed assistance, and the different paths students take to learning in individual and collaborative conditions. Then, we described how CTRL could be used as a basis for two different examples of adaptive collaborative learning support. We outlined the limitations of CTRL with respect to defining conditions of interaction, and discussed how while CTRL as a concept is simple to adopt, the complexity lies in refactoring existing systems to create the necessary components.

We see one of the main contributions of our work as the development of a framework that supports the integration of pre-existing and custom-built components, with a particular focus on tutoring components. Using CTRL, we combined a pre-existing domain model of a tutee's problem-solving performance with a collaborative model of a peer tutor's correction actions, and delivered feedback that included both a prompt to collaborate and a domain hint. There are difficulties to relying heavily on existing tutoring systems for components, because it may be necessary to refactor the components or deal with legacy code that is difficult to appropriate for new purposes. However, in implementing our three experimental conditions, we leveraged CTA logging protocols, interface components, and cognitive models, which would have been 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 substituted peer tutoring for cognitive tutoring and collaborative domain support for individual domain support, but we did not explore collaborative scenarios that would involve tutoring or forms of collaborative support other than collaborative domain support. In our view, remaining close to the CTA was the first natural step. We plan to tackle further extensions 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 understand what the student knows and does not know). As we develop more components, they will form a basis for the construction of more general collaborative scenarios. Further, using CTRL, it will eventually be possible to apply our domain-general collaborative components to provide collaborative tutoring for other tasks with pre-existing domain models. The other main contribution of our framework is that it makes it easier to implement comparison conditions, by placing the integration logic in a control module. 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 peer tutoring condition. Traditionally, these scenarios would have been implemented in very different manners, rather than using the same architectural framework. Furthermore, adding a new scenario to function as a comparison condition took very little effort once the scenario components had been implemented. One limitation of CTRL is that currently

only simple integration scenarios are accommodated; it has not yet been tested with more complex configurations. As the conditions that we attempt to implement evolve and become more complex, so will the framework.

CTRL, the collaborative tutoring research lab, is an initial step toward supporting research into complex forms of adaptive assistance for collaborative learning. There have generally between two types of work in this area: Research that attempts to understand from an educational psychology perspective whether and how adaptive assistance can be effective to promote collaborative learning, and research that attempts to understand from a technological perspective how to construct models of collaboration and provide automated adaptive assistance. In the first case, educational psychologists often lack the technological tools required to implement adaptive systems, and thus conduct wizard-of-oz studies or work with programmers to implement technologically less than optimal interventions. On the other hand, technologists focus their energies on determining how to create complex systems, but the output is often a research prototype that is not generally evaluated to determine its effect on student collaboration and learning. What we offer in this paper is a way to bridge the gap between the two approaches, making it easier to move from implementing adaptive systems to evaluating them, and iterate upon existing adaptive systems to improve the quality of the support that they can provide. We believe that such a bridge is necessary in order to create adaptive systems that can have a real impact on classrooms; it does not matter if impressive adaptive systems are being developed if they do not have a positive effect on collaboration and learning, and psychology experiments may develop a restricted theory of adaptive assistance if they only experiment with suboptimal, low-tech solutions. It is our hope that the structure of CTRL, and in particular its integration framework, facilitates more complex forms of support by leveraging domain-specific models, a more controlled evaluation by allowing the construction of comparison conditions using pre-existing components, and iteration on the development of adaptive support.

Acknowledgements This research was supported by the Pittsburgh Science of Learning Center, NSF Grant # 0354420. We thank Bruce McLaren for his valuable contributions during initial stages of the project. Thanks to Jonathan Steinhart, Dale Walters, and Steve Ritter for their support concerning the use of the Carnegie Learning Cognitive Tutor Algebra code, and to Ido Jamar, Kathy Dickensheets, and the teachers from CWCTC for their motivated involvement in the project. Finally, thanks to Carolyn Rosé, Dejana Diziol and Amy Ogan for their comments.

References

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

Aleven, V., McLaren, B., Roll, I., Koedinger, K.: Toward meta-cognitive tutoring: a model of help seeking with a Cognitive Tutor. Int. J. Artif. Intell. Educ. **16**, 101–128 (2006)

Anderson, J.R., Pelletier, R.: A development system for model-tracing tutors. In: Birnbaum, L. (eds.) Proceedings of the 1991 International Conference of the Learning Sciences, pp. 1–8. Charlottesville, AACE, VA (1991)

Azevedo, R.: Computer environments as metacognitive tools for enhancing learning. Educ. Psychol. **40**(4), 193–197 (2005)

Baghaei, N., Mitrovic, A., Irwin, W.: Supporting collaborative learning and problem solving in a constraint-based CSCL environment for UML class diagrams. Int. J. Comput. Support. Collab. Learn. **2**(2–3), 159–190 (2007)

Baker, R.S.J.d., Corbett, A.T., Koedinger, K.R., Evenson, S., Roll, I., Wagner, A.Z., Naim, M., Raspat, J., Baker, D.J., Beck, J.: Adapting to when students game an intelligent tutoring system. In: Ikeda, M., Ashley, K., Chan, T.W. (eds.) Proceedings of 8th International Conference on Intelligent Tutoring Systems, pp. 392–401. Springer Verlag, Berlin (2006)

Beck, J.E., Mostow, J., Bey, J.: Can automated questions scaffold children's reading comprehension? In: Lester, J.C., Vicari R.M., Paraguacu, F. (eds.) Proceedings of 7th International Conference on Intelligent Tutoring Systems, pp. 478–490. Springer Verlag, Berlin (2004)

Berge, O., Slotta, J.: Learning technology standards and inquiry-based learning. In: Koohang, A., Harman, K. (eds.) Learning Objects and Instructional Design, pp. 327–358. Informing Science Press, Santa Rosa (2007)

Biswas, G., Leelawong, K., Schwartz, D., Vye, N.: The Teachable Agents Group at Vanderbilt: Learning by teaching: a new agent paradigm for educational software. Appl. Artif. Intell. **19**(3), 363–392 (2005)

Carnegie learning: Carnegie learning, Inc. Math Curricula with Proven Success. Retrieved 1 Oct 2009 from http://www.carnegielearning.com/ (2009)

Constantino-González, M.A., Suthers, D., Escamilla de los Santos, J.: Coaching web-based collaborative learning based on problem solution differences and participation. Int. J. Artif. Intell. Educ. **13**(2–4), 263–299 (2003)

Del Solato, T., du Boulay, B.: Formalization and implementation of motivational tactics in tutoring systems. Int. J. Artif. Intell. Educ. **6**(4), 337–378 (1995)

Dillenbourg, P., Baker, M., Blaye, A., O'Malley, C.: The evolution of research on collaborative learning. In: Spada, H., Reimann, P. (eds.) Learning in Humans and Machine: Towards an Interdisciplinary Learning Science, pp. 189–211. Elsevier, Oxford (1995)

Dillenbourg, P., Hong, F.: The mechanics of CSCL macro scripts. Int. J. Comput. Support. Collab. Learn. **3**(1), 5–23 (2008)

Dillenbourg, P.: 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? pp. 61–91, Open Universiteit Nederland, Heerlen (2002)

Diziol, D., Rummel, N., Kahrimanis, G., Guevara, T., Holz, J., Spada, H., Fiotakis, G.: Using contrasting cases to better understand the relationship between students' interactions and their learning outcome. In: Kanselaar, G., Jonker, V., Kirschner, P.A., Prins, F. (eds.) Proceedings of the Eighth International Conference of the Learning Sciences, pp. 348–349. International Society of the Learning Sciences, Inc., Utrecht (2008)

Fantuzzo, J.W., Riggio, R.E., Connelly, S., Dimeff, L.A.: Effects of reciprocal peer tutoring on academic achievement and psychological adjustment: a component analysis. J. Educ. Psychol. **81**(2), 173–177 (1989)

Fischer, F., Kollar, I., Mandl, H., Haake , J.M. (eds.): *Scripting Computer-Supported Collaborative Learning—Cognitive, Computational, and Educational Perspectives* (Springer, New York 2007)

Fuchs, L., Fuchs, D., Hamlett, C.L., Phillips, N.B., Karns, K., Dutka, S.: Enhancing students' helping behavior during peer-mediated instruction with conceptual mathematical explanations. Elem. Sch. J. **97**(3), 223–250 (1997)

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

Große, C.S., Renkl, A.: Finding and fixing errors in worked examples: can this foster learning outcomes'? Learn. Instr. **17**, 612–634 (2007)

Gweon, G., Rose, C., Carey, R., Zaiss, Z.: Providing support for adaptive scripting in an on-line collaborative learning environment. In: Grinter, R., Rodden, T., Aoki, P., Cutrell, E., Jeffries, R., Olson, G. (eds.) Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems, pp. 251–260. ACM Press, New Jersey (2006)

Hmelo-Silver, C.E.: Problem-based learning: what and how do students learn? Educ. Psychol. Rev. **16**(3), 235–266 (2004)

Hoppe, H.U.: The use of multiple student modeling to parameterize group learning. In: Greer, J. (ed.) Proceedings of the 7th International Conference on Artificial Intelligence in Education, pp. 234–241. Association for the Advancement of Computing in Education (AACE), Charlottesville, VA (1995)

Hoppe, H.U., Gaßner, K.: Integrating collaborative concept mapping tools with group memory and retrieval functions. In: Stahl, G. (ed.) Proceedings of CSCL 2002, pp. 115–124. Lawrence Erlbaum, Boulder, CO (2002)

Israel, J., Aiken, R.: Supporting collaborative learning with an intelligent web-based system. Int. J. Artif. Intell. Educ. **17**(1), 3–40 (2007)
Koedinger, K.R., Aleven, V.: Exploring the assistance dilemma in experiments with Cognitive Tutors. Educ. Psychol. Rev. **19**(3), 239–264 (2007)
Koedinger, K., Anderson, J., Hadley, W., Mark, M.: Intelligent tutoring goes to school in the big city. Int. J. Artif. Intell. Educ. **8**, 30–43 (1997)
Koedinger, K., Corbett, A.T.: Cognitive tutors: technology bringing learning science to the classroom. In: Sawyer, R.K. (ed.) The Cambridge Handbook of the Learning Sciences, pp. 61–78. Cambridge University Press, New York (2006)
Krueger, C.W.: Software reuse. ACM Comput. Surv. **24**(2), 131–183 (1992)
Kumar, R., Rosé, C.P., Wang, Y.C., Joshi, M., Robinson, A.: Tutorial dialogue as adaptive collaborative learning support. In: Luckin, R., Koedinger, K.R., Greer, J. (eds.) Proceedings of the 13th International Conference on Artificial Intelligence in Education, pp. 383–390. IOS Press, Amsterdam (2007)
McManus, M.M., Aiken, R.M.: Teaching collaborative skills with a group leader tutor. Educ. Inf. Technol. **1**, 75–96 (1996)
McNamara, D.S., O'Reilly, T., Rowe, M., Boonthum, C., Levinstein, I.B.: iSTART: a web-based tutor that teaches self-explanation and metacognitive reading strategies. In: McNamara, D.S. (ed.) Reading Comprehension Strategies: Theories, Interventions, and Technologies, pp. 397–421. Erlbaum, Mahwah (2007)
Medway, F.J., Baron, R.M.: Locus of control and tutor's instructional style. Contemp. Educ. Psychol. **2**, 298–310 (1997)
Meier, A., Spada, H., Rummel, N.: A rating scheme for assessing the quality of computer-supported collaboration processes. Int. J. Comput. Supp. Collab. Learn. **2**, 63–86 (2007)
Mostow, J., Aist, G.: Evaluating tutors that listen: an overview of Project LISTEN. In: Forbus, K., Feltovich, P. (eds.) Smart Machines in Education, pp. 169–234. MIT/AAAI Press, Menlo Park (2001)
Mühlenbrock, M., Tewissen, F., Hoppe, H.U.: A framework system for intelligent support in open distributed learning environments. Int. J. Artif. Intell. Educ. **9**, 256–274 (1998)
Mühlenbrock, M.: Shared workspaces: analyzing user activity and group interaction. In: Hoppe, H.U., Ikeda, M., Ogata, H. (eds.) New Technologies for Collaborative Learning, Kluwer Academic Publishers, Dordrecht (2004)
Pfister, H.R., Mühlpfordt, M.: Supporting discourse in a synchronous learning environment: The learning protocol approach. In: Stahl, G. (ed.) Proceedings of CSCL 2002, pp. 581–589. Erlbaum, Hillsdale, NJ (2002)
Pinkwart, N.: A plug-in architecture for graph based collaborative modeling systems. In: Hoppe, U., Verdejo, F., Kay, J. (eds.), Proceedings of the 11th International Conference on Artificial Intelligence in Education, pp. 535–536. IOS Press, Amsterdam (2003)
PSLC: Welcome: Pittsburgh Science of Learning Center. Retrieved 1 Oct 2009 from http://www.learnlab.org/ (2009)
Ritter, S., Blessing, S.B., Hadley, W.S.: SBIR phase I final report 2002. Department of Education RFP ED: 84-305S (2002)
Ritter, S., Koedinger, K.R.: An architecture for plug-in tutor agents. Int. J. Artif. Intell. Educ. **7**(3/4), 315–347 (1996)
Rosatelli, M., Self, J.: A collaborative case study system for distance learning'. Int. J. Artif. Intell. Educ. **14**(1), 97–125 (2004)
Roscoe, R.D., Chi, M.: Understanding tutor learning: knowledge-building and knowledge-telling in peer tutors' explanations and questions. Rev. Educ. Res. **77**(4), 534–574 (2007a)
Roscoe, R.D., Chi, M.: Tutor learning: the role of instructional explaining and responding to questions. Instr. Sci. **36**(4), 321–350 (2007b)
Rummel, N., Spada, H.: Learning to collaborate: an instructional approach to promoting collaborative problem-solving in computer-mediated settings. J. Learn. Sci. **14**(2), 201–241 (2005)
Saab, N., van Joolingen, W.R., van Hout-Wolters, B.H.A.M.: Supporting communication in a collaborative discovery learning environment: the effect of instruction. Instr. Sci. **35**(1), 73–98 (2007)
Strijbos, J.W., Martens, R.L., Jochems, W.M.G.: Designing for interaction: six steps to designing computer-supported group-based learning. Comput. Educ. **42**(4), 403–424 (2004)
Slavin, R.E.: Research on cooperative learning and achievement: What we know, what we need to know. Contemp. Educ. Psychol. **21**(1), 43–69 (1996)

Soller, A., Martinez, A., Jermann, P., Mühlenbrock, M.: From mirroring to guiding: a review of state of the art technology for supporting collaborative learning. Int. J. Artif. Intell. Educ. **15**(4), 261–290 (2005)
Soller, A.: Computational modeling and analysis of knowledge sharing in collaborative distance learning. User Model. User Adapt. Interact. J. Pers. Res. **14**(4), 351–381 (2004)
Suebnukarn, S., Haddawy, P.: A collaborative intelligent tutoring system for medical problem-based learning. In: Nuno, N., Charles, R. (eds.) Proceedings of the 9th International Conference on Intelligent User Interfaces, pp. 14–21. ACM Press, New York, NY (2004)
Suthers, D.: Architectures for computer supported collaborative learning. In: Okamoto, T., Hartley, R., Klus, Kinshuk, J.P. (eds.) Proceedings of the IEEE International Conference on Advanced Learning Technology: Issues, Achievements and Challenges, pp. 25–28. IEEE Computer Society, Los Alamitos, CA (2001)
Tedesco, P.: MArCo: building an artificial conflict mediator to support group planning interactions. Int. J. Artif. Intell. Educ. **13**(1), 117–155 (2003)
Tsovaltzi, D., Rummel, N., Pinkwart, N., Scheuer, O., Harrer, A., Braun, I. McLaren, B.M.: CoChemEx: supporting conceptual chemistry learning via computer-mediated collaboration scripts. In: Dillenbourg, P., Specht, M. (eds.) Proceedings of the Third European Conference on Technology Enhanced Learning, pp. 437–448. Springer, Berlin (2008)
VanLehn, K.: The behavior of tutoring systems. Int. J. Artif. Intell. Educ. **16**(3), 227–265 (2006)
VanLehn, K., Koedinger, K. R., Skogsholm, A., Nwaigwe, A., Hausmann, R. G. M., Weinstein, A., Billings, B.: What's in a step? Toward general, abstract representations of tutoring system log data. In: Conati, C., McCoy, K.F., Paliouras, G. (eds.) User Modeling 2007, 11th International Conference, pp. 455–459. Springer (2007)
Vieira, A. C., Teixeira, L., Timóteo, A., Tedesco, P., Barros, F. A.: Analyzing on-line collaborative dialogues: The OXEnTCHÊ-Chat. In: Lester, J.C., Vicari, R.M., Paraguaçu, F. (eds.) Proceedings of the 7th International Conference on Intelligent Tutoring Systems, pp. 315–324. Springer-Verlag, Germany (2004)
Vizcaíno, A., Contreras, J., Favela, J., Prieto, M.: An adaptive collaborative environment to develop good habits in programming. In: Gauthier, G., Frasson, C., VanLehn, K. (eds.) 5th International Conference on Intelligent Tutoring Systems, ITS'2000, pp. 262–271. Springer-Verlag, Berlin (2000)
Walker, E., Rummel, N., Koedinger, K.: To tutor the tutor: adaptive domain support for peer tutoring. In: Woolf, B., Aimeur, E. Nkambou, R., Lajoie, S. (eds.) Proceedings of the 9th International Conference on Intelligent Tutoring Systems, pp. 626–635. Springer, Berlin (2008)
Walker, E., Rummel, N., McLaren, B., Koedinger, K.: The student becomes the master: Integrating peer tutoring with cognitive tutoring. In: Chinn, C.A., Erkens, G., Puntambekar, S. (eds.) Proceedings of the Computer Supported Collaborative Learning (CSCL) Conference 2007, pp. 750–752. International Society of the Learning Sciences, New Brunswick (2007)

Author Biographies

Erin Walker is a Ph.D. candidate in Human-Computer Interaction at Carnegie Mellon University. She received her BS with honors in both Computer Science and Psychology from the University of Manitoba in 2004. Her primary research interests lie in the area of adaptive collaborative learning support, with three emphases: how to design the support to have a positive influence on student behavior, how to efficiently implement the support using existing problem-solving models, and what effects the support has on domain learning outcomes.

Nikol Rummel is an Assistant Professor in the Institute of Psychology at the University of Freiburg, Germany, and an Adjunct Assistant Professor in the Human-Computer Interaction Institute at Carnegie Mellon University. Dr. Rummel received a Diploma and a Ph.D. in psychology from the University of Freiburg and a M.Sc. degree in educational psychology from the University of Wisconsin—Madison, USA as a Fulbright scholar. Before her training in Psychology, she completed two years of teacher training at the Pädagogische Hochschule (College of Education) Freiburg. Dr. Rummel's research interests center around issues of instructional support for learning in computer-supported settings. The focus of recent work at the Pittsburgh Science of Learning Center was on adaptive support for collaborative learning with the Cognitive Tutor Algebra.

Kenneth R. Koedinger is Professor of Human-Computer Interaction and Psychology at Carnegie Mellon. His background includes a BS in Mathematics, a MS in Computer Science, a Ph.D. in Cognitive Psychology, and experience teaching in an urban high school. His research has contributed new principles and techniques for the design of educational software and has produced basic cognitive science research results on the nature of mathematical thinking and learning. Dr. Koedinger directs the Pittsburgh Science of Learning Center and is co-founder of Carnegie Learning Inc., a company bringing learning research to schools in the form of the Cognitive Tutor technology.