Architecture for building Conversational Agents that support Collaborative Learning

Rohit Kumar and Carolyn P. Rosé

Abstract— Tutorial Dialog Systems that employ Conversational Agents to deliver instructional content to learners in one-on-one tutoring settings have been shown to be effective in multiple learning domains by multiple research groups. Our work focuses on extending this successful learning technology to collaborative learning settings involving two or more learners interacting with one or more agents. Experience from extending existing techniques for developing conversational agents into multiple-learner settings highlights two underlying assumptions from the one-learner setting that do not generalize well to the multi-user setting, and thus cause difficulties. These assumptions include what we refer to as the near-even participation assumption and the known addressee assumption. A new software architecture called Basilica that allows us to address and overcome these limitations is a major contribution of this article. The Basilica architecture adopts an object-oriented approach to represent agents as a network composed of what we refer to as behavioral components because they enable the agents to engage in rich conversational behaviors. Additionally, we describe three specific conversational agents built using Basilica in order to illustrate the desirable properties of this new architecture.

Index Terms— Collaborative learning, Intelligent agents, Natural language interfaces, Software Architectures

1 INTRODUCTION

IN this article we present Basilica, which is a novel architecture and tool kit for utilizing Conversational Agent (CA) technology to support collaborative learning in a powerful way. In our previous publications [1] [2] [3] [4] [5] [6], we have demonstrated substantial learning gains, sometimes with effect sizes equivalent to 1.24 letter grades, in connection with dynamic support for collaborative learning that has been enabled through this architecture in comparison with no support or static support control conditions. This article is the first, comprehensive discussion of the architecture that transcends individual instantiations, and explains how the architecture can be adapted and used by researchers and practitioners alike for their own system development.

Applications of Conversational Agent (CA) technology for automated tutoring have been extensively researched, particularly in the last 15 years. Various research groups have developed agents in a variety of domains including reading, algebra, geometry, calculus, computer literacy, physics, programming, foreign languages, research methods and thermodynamics. Many evaluations have shown that CAs can be effective tutors [1] [7] [8]. The dynamic nature of dialogue offers the flexibility of supporting students, using feedback to scaffold their process in a personalized way, specifically where it is needed. One can think of them as agents that elicit explanations in a step by step fashion with each step of the line of reasoning contributing to the resulting explanation. Although explanation activities have intrinsic cognitive benefits and students often learn from explanation activities without feedback, such as unsupported self-explanation during problem solving or worked example studying, many students provide explanations of low quality in these contexts [9]. Feedback on explanation quality should support the generation of higher-quality explanations. Furthermore, in computer-based learning environments, such feedback is likely to provide a crucial incentive for students to take the explanation task seriously [10].

Most state of the art CAs for automated tutoring applications have been developed and evaluated for one-on-one tutoring situations where the agent interacts with one learner at a time. In our research, we have focused on extending the success of CAs in tutoring applications to collaborative learning situations where the agents interact with multiple learners who are part of a collaborative team working through a learning task. This approach grows out of the literature on script-based support for collaborative learning. In order to encourage productive patterns of collaborative discourse, researchers both in the

- *Rohit Kumar is with the Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: rohitk@ cs.cmu.edu.*
- *Carolyn P. Rosé is with the Language Technologies Institute and the Human Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: cprose@ cs.cmu.edu.*

Manuscript received (April 14, 2010), revised (August 16, 2010)

Computer Supported Collaborative Learning (CSCL) tradition and the Educational Psychology tradition have separately developed approaches to using scripts for scaffolding the interactions between students, to help coordinate their communication, and to encourage deep thinking and reflection [11]. Script based support has traditionally taken on a wide range of forms including provision of prompts during collaboration [12] and design of structured interfaces including such things as buttons associated with typical "conversation openings" [13]. Previous approaches to scripting have been static, one-size-fits-all approaches. In other words, the approaches were not responsive to what was happening in the collaboration. This non-adaptive approach can lead to over scripting [14] or interference between different types of scripts [15]. With these things in mind, and considering that ideally we would like students to internalize the principles encoded in the script based support, a more dynamic and potentially more desirable approach would be to trigger support based on observed need and to fade scaffolding over time as students acquire the skills needed to collaborate productively in a learning context, as is enabled through automatic collaborative learning process analysis [16]. We have conducted a series of studies over the past 3 years in which we have successfully evaluated CA technology as a form of dynamic support for collaborative learning achieving improvements in learning of a full letter grade or more in comparison with no support and static support control conditions.

Collaborative learning settings are a subset of the general class of multi-party interactive situations, which presents challenges beyond those posed in the more typical single agent / single user settings where much of the CA research to date has focused. Research on CAs and dialog systems in single user applications has explored representations and implementations that can be extended to agents in the multi-party case, but with difficulty. Addressing this difficulty is the focus of the research presented in this article, which not only supports the important goal of enabling the efficient development of highly effective computer-supported collaborative learning environments, but also paves the way for a broader class of applications of CAs in other types of collaborative environments.

In the remainder of the paper we first discuss specific assumptions underlying typical state based and plan-based approaches to CAs that pose problems for the multi-party scenarios in which we use CAs in our work. This discussion will motivate the specific approach we take in our work. Section 3 discusses the technical details of the Basilica Architecture and describes the agent representation using a simple example. Section 4 describes a collection of agents that support teams of learners in a variety of collaborative learning tasks in order to demonstrate the capabilities of the Basilica architecture. This is followed by discussion and directions for future work in section 5.

2 Motivation

The design of Basilica grew from three primary desiderata, namely, a rich enough representation to enable the expression of the types of rich behaviors we envision, a structure that is modular enough to accommodate the type of complex conversational dynamics that arise in multi-party settings, and finally an approach that supports reuse and mix-and-match construction in order to enable building complex behaviors from the interaction of simpler behaviors so that the amount of effort to develop new systems in Basilica reduces over time.

2.1 Rich Representational Capability

To achieve autonomous behavior by agents, characterized as involving a combination of simulated cognition and control, it was important to achieve a level of representational richness that allows us to model the agent in the concerned general class of conversational situations. However, while it would be relatively simple to add complexity to the representation if that was the only consideration, it could easily lead to the downside that the effort involved in authoring (or programming) the knowledge and the procedures that enable the agent to participate in specific situations would increase beyond what is practical. Thus, the consideration of representational adequacy and efficiency of implementation often conflict with each other. For example: A simple representation like a finite state machine [17] [18] is suitable only for relatively short and simple interactions, but the development effort involved is relatively small and is often facilitated using state-machine authoring tools [19]. Richer representation like the ones used in plan-based approaches [20] [21] model the conversational goal(s) of the agent and use planning algorithms to determine a sequence of steps that can achieve the goal(s). While such approaches have been shown to be flexible and robust in conversational situations like mixed-initiative dialog, a considerable amount of effort is involved in specifying the goal representations, operators, potential steps, pre-conditions, etc., required by the underlying planning algorithms.

In the new architecture proposed here, we adopt a rich representational capability that is not restricted by a small set of interaction operators. This enables the developers of CAs in collaborative learning situations to program complex interactive behaviors like the ones we describe in the case-studies in Section 4. At the same time, it is possible to use existing representations that reduce development effort as discussed in Section 3.3.

2.2 Flexibility to address Complex Interaction Dynamics

Considering the amount of research that has gone into developing approaches for building CAs that are capable of conversational interaction with one user in each session, it is natural to push the envelope in order to deploy agents in multi-user interactive situations. However, typical approaches to developing CAs for single user settings make heavy use of the simplifying assumption that there are only two participants in the interaction, namely the human user and the agent. From this fundamental assumption come two more practical assumptions. One is that there will be a relatively even participation of both parties, which will typically mean that speakers take turns alternately. And the other assumption is the known addressee assumption, namely that if there are only two participants, then the addressee must always be the one who is not the speaker. Here we discuss why these assumptions break down in multi-party scenarios and what practical implications that has.

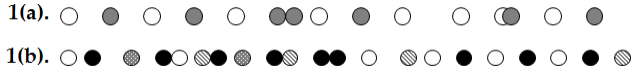


Fig. 1. Failure of Even Participation assumption in Two-Party Interaction 1(a) and Multi-party interaction 1(b)

Fig. 1a represents a typical interaction in a single user (two-party) scenario. The white dots represent the agent turns and the grey dots represent the user turns. Notice that typically white and grey alternate with one another. This is not the case 100% of time. Nevertheless, it is true often enough that if the system behaves in a way that presupposes that this will always be the case, it won't make mistakes very often. In this specific example, at turns 6 & 7, the user is responding to a tutor prompt for information. After the user has provided an initial answer, he then provides additional information (or a correction) to the system. However, since the agent (system) is based on an even participation assumption, at turn 8, the system is still trying to respond to turn 6 from the user and ignores the information provided in turn 7. In the last few years, work on flexible turn taking [22] [23] has proposed sophisticated models that can help the system anticipate the possibility of failure of such an assumption and avoid (or recover from) potential failure in the interaction. However, as illustrated in Fig. 1b, in the case of a multi-party interaction, the failure of even-participation assumption is not an exception to be recovered from. Instead it is a normal characteristic of the dynamic interaction in multi-party settings. In this example, each color represents a different speaker. As can be seen, speakers do not alternate in any predictable pattern of even participation. Assuming the white dots represent the agent turns, we can see that ambiguity about which contribution to consider as an answer to its prompts is common rather than a rare exception.

Related to the problem just described is the problem of knowing who the addressee of an utterance is. When there are multiple speakers, sometimes the speakers will be talking to each other, and not the agent. Mainly the agent needs to know when it is being addressed, but this is far from trivial in this multi-user case. The known addressee assumption on which conversational agents for two-party interactions are developed implies in the two-party case that the addressee is the other speaker. A naïve extension of this assumption to the multi-party case would be that contributions from each participant are addressed to all the other participants, which includes the agent. Failure of this assumption happens when the user says something that is not addressed to the agent, or even if the agent is among the users addressed, but the agent's prompts are not addressed. This is illustrated in the excerpt from a thermodynamics tutorial dialog system shown in Table 1.

TABLE 1
EXCERPT OF A CONVERSATION BETWEEN A TUTOR AND TWO STUDENTS

Student1	OK, lets start
Tutor	What would happen to the power output of a Rankine Cycle at a higher operating temperature?
Student2	hmmm … Can you answer that?
Student1	I think it will increase.
Tutor	The correct answer is that at a higher operating temperature, more heat is added to the cycle and hence the power output increases too. What about the heat rejected by the cycle though?
Student1	You are right S1. It increases too. Lets move on to the next topic.

When Student2 asks Student1 to respond to the tutor's first question, the tutor follows the known addressee assumption and considers Student2's turn as a response to its previous question. The response is evaluated as an incorrect (or non-understandable) answer and the tutor provides the correct answer. Meanwhile, Student1's response to the first question is considered as a response to the second question from the tutor.

State-of-the-art conversational agents implement error recovery strategies [24] designed to deal with non-understandings or misunderstandings in order to recover from local failures of the known addressee assumption. In multi-party scenarios, the dynamics of responding to a turn from the user becomes increasingly complicated and task specific. For example, in a collaborative learning set-

ting, students may choose to discuss the answer to a tutor turn among themselves before responding to the tutor.

An additional complicating factor is the duration of interaction with agents in typical collaborative learning scenarios. Agents that interact with one user at a time have been developed for interactive situations that do not require any more than a few minutes of interaction. As we extend the application of these agents to a whole learning session, which could last from 30 minutes to multiple hours, the structure of the conversation becomes increasingly complex, and the breakdown of the above assumptions become increasing likely.

Failure of the two problematic assumptions described in this section prevents generalization of the agent support shown to be effective for one learner to situations involving several learners. Other approaches explored in CSCL research bypass the inability to participate in this increased level of interaction by structuring the interaction between learners in various ways. For example by assigning tutoring roles to the learners [25] allows the agents to focus on supporting the role of only one learner. Other work [13] has explored design of interactive environments that encourage the students to perform communicative acts by using a highly structured communication modality. However, the use of such modalities reduces the interactivity and restricts social interaction between students which could be detrimental over long term interaction [26].

Group dynamics is another consideration. As the number of students participating in the interaction increases, the exchange between them becomes a bigger factor in the maintaining the quality of interaction. Thus, agents may have to monitor and regulate the interaction between students in order to provide appropriate support for the learning in a group setting [27]. For example, in an extended collaborative learning interaction, the tutor may elicit participation from students who are contributing less than other students. Such situation specific interaction dynamics may be more complex than the scenarios that have been explored in current work [28].

The proposed architecture provides the flexibility to develop conversational agents that can implement interaction strategies (and tactics) that would enable them to participate in multi-party interactive situations.

2.3 Reducing Development Effort

While the primary motivations that guided the design of our new architecture was to increase the representational capability that encodes the knowledge and the behavior of the agent, as well as implement agents capable of performing sophisticated interaction tactics and strategies, we note that these improvements could easily increase the effort involved in developing the conversational agents. Thus, a final objective of our effort has been to develop the architecture in a way that reduces development effort. To alleviate additional effort to some extent, the architecture adopts principles of object-oriented design. Modeling the agent as a collection of appropriately sized, independent objects allows incremental development as well as reusability as discussed in the next section. Earlier related work on architectures for conversational agents [29] has also employed similar object-oriented programming principles for developing conversational agents.

3 Proposal

3.1 A Model of Interaction

Before we proceed to discuss the details of the architecture for building conversational agents proposed in this paper, we present a model of interaction between a conversational participant and the environment. The participant could be a human user or an agent and the environment includes other participants and observers. The environment also specifies the modalities of interaction (e.g., text, voice, video, gesture, etc.) based on its affordances.

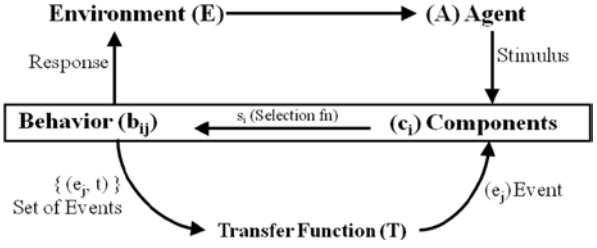


Fig. 2. A behaviorist model for Conversational Agents

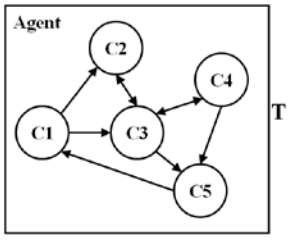
In the model of interaction shown in Fig. 2, the participant (Agent) observes stimuli from the environment (like entrance of a new participant, action by one of the current participants, change in environment like server notifications, etc.). These stimuli are conveyed to the perception components of the agent, which process the stimuli in order to determine whether any relevant behavior is to be triggered to process the stimuli. For example, in a telephone based interaction, the listener component (ears) triggers the behavior of hearing upon receiving the voice stimulus from the handset (medium/environment). The triggered behavior may respond by generating events (internal to the agent) as well as by sending a response back to the environment. The generated events are transferred to other components. This way the environmental stimulus is propagated through a collection of components that implement all the singular behaviors an agent can perform. As a result of the stimulus propagation, a response may be sent back to the environment, and internal states of each component may be updated.

3.2 Basilica: The new architecture[1]

The Basilica architecture is based on the above described model of interaction between conversational participants and the environment. Agents built using the Basilica architecture are implemented as a collection of what we refer to as behavioral components. Computational capabilities like perception, action, cognition, affect and memory are implemented as behaviors. Commonly used components corresponding to these behaviors are discussed in section 4.4. The selection function (s_i) for each component (c_i) is implemented as a one to one mapping function that maps the type of event / stimulus to behavior (b_{ij}). Each behavior is programmatically defined as a function that responds to a type of event by generating a set of zero or more events.

$$b_{ij}\left(type\left[event_k\right]\right) \rightarrow \left\{events\right\} \qquad (1)$$

The transfer function (T) is specified by a network of components. Events generated by a behavior b_{ij} are propagated to all components that are connected to component c_i. The connections of the network are unidirectional, i.e., if component c_1 can receive events from c_2, then c_2 does not necessarily receive events from c_1 (unless so connected). For example, in the network shown in Fig. 3, only components c_2 and c_3 receive events generated by c_1.



T = { (C1 ,C2), (C1, C3),
(C2, C3),
(C3, C2), (C3, C4), (C3, C5),
(C4, C3), (C4, C5),
(C5, C1) }

Fig. 3. Example of an Agent's Component Network (T)

The Basilica architecture provides a set of core abstract classes (implemented in Java) for defining agents, components, connections and events. While the behaviors performed by the agent are specific to the agent's implementation and change between agents, the Basilica architecture provides low-level functionality required to implement the agents.

Foremost within this scope is the control mechanism for propagating events between components. Events are propagated as a broadcast to all connected components as discussed above. While sometimes this might cause components to receive events that they do not need to process, the broadcast mechanism allows for relatively simpler specification of the network. Additionally, the architecture provides developers the ability to selectively transmit events to a subset of all connected components. Basilica is responsible for initializing and maintaining the connection between components over which events are transmitted. Besides maintaining these connections, the architecture provides observer interfaces that allow developers to observe events as they are transmitted over the connections. This supports creation of graphical displays that can be used by facilitators and moderators. The debugging interface discussed in section 3.5 uses this mechanism.

Second, the abstract classes used for defining behavioral components provide a generic mechanism for initializing, executing and observing each component. By default this mechanism allows each component to perform its behaviors asynchronously by executing each component in a separate thread. So, if a particular component (like a parser) takes an extended amount of time for processing its events, the other components are not blocked from processing their events.

Third, the selection function (s_i) within each component is responsible for accumulating incoming events and triggering their corresponding behaviors (b_{ij}). Basilica implements a generic mechanism for this function. By default, events are buffered and processed sequentially in the order in which events are received. However, the object oriented implementation of Basilica allows developers to override this default mechanism for special purpose components to prioritize certain kinds of stimuli (like a user barge-in or a resource unavailability notification).

Besides the core classes, Basilica provides a generic class for a memory component (described in Section 4.2) that provides the ability to keep state-based information accessible across components. Additionally, the architecture provides an agent factory class that allows runtime agent construction from an XML specification like the one shown in Fig. 4.

3.3 TuTalk: Integrating existing behavior within Agents

Basilica allows for integration of a wide range of behavioral components, but one that we have used frequently in our agents is the TuTalk dialog engine [30]. In the next section we will discuss its role within an example Basilica agent, illustrated in Fig. 6. TuTalk is a state-based dialogue engine that operates using what are referred to as tutoring scripts. Tutoring scripts compatible with the TuTalk dialog engine define directed lines of reasoning composed of a sequence of steps that implement an Initiation – Response - Feedback interaction pattern with the goal of leading a student to construct a correct explanation for a complex concept as independently as possible. The dialog engine executes these steps by presenting the Initiation question, matching the student response and

[1] Researchers interested in using this architecture may contact the first author. Updates and other information about the architecture are aggregated at http://basilica.rohitkumar.net/wiki/

```
<agent name="Tutor">
    <components>
        <component name="SLListener"        class="agent.components.SLListener"/>
        <component name="MessageFilter"     class="agent.components.MessageFilter"/>
        <component name="TouchFilter"       class="agent.components.TouchFilter"/>
        ...
        ...
        <component name="OutputCoordinator" class="agent.components.OutputCoordinator"/>
        <component name="SLActor"           class="agent.components.SLActor"/>
    </components>
    <connections>
        <connection from="SLListener"       to="MessageFilter"/>
        <connection from="SLListener"       to="TouchFilter"/>
        <connection from="TouchFilter"      to="GreetingActor"/>
        ...
        ...
        <connection from="TutoringActor" to="OutputCoordinator"/>
        <connection from="OutputCoordinator" to="SLActor"/>
    </connections>
</agent>
```

Fig. 4. Example XML Specification of a Basilica Agent

presenting appropriate feedback before moving on to the next step. The script formalism also allows introducing another intervening sequence of remedial steps as feedback to incorrect responses. Thus, support is provided on an as-needed basis. In order to facilitate authoring of these scripts, TuTalk provides a set of authoring tools for rapid development of these scripts by subject matter experts who may not be technology experts.

Integration of TuTalk within Basilica's tutoring components demonstrates the flexibility to integrate existing tools and interactive representations within agents built using this architecture. Tables 1 and 2 illustrate example interactions with authored TuTalk agents. Note that the TuTalk dialog engine inherently does not provide a mechanism to address the issues related to multi-party interaction discussed earlier. However, Basilica allows us to augment these tutoring components with other necessary behavior to address the issues related to complex interaction dynamics without needing to add any sophistication to component technologies themselves.

Fig. 5. Two students interacting with the Second Life Tutor

3.4 An Example Agent: SecondLife Tutor

Now we will demonstrate how an agent built using the Basilica architecture works using an example agent that tutors a team of students in the SecondLife (SL) virtual environment. The SecondLife Tutor shown in Fig. 5 is implemented as a SecondLife object (seen as the spherical object in the figure).

The SecondLife tutor performs two types of user observable behaviors, i.e., greeting and tutoring. To customize the tutoring behavior, the tutor can be augmented with a list of TuTalk scripts and the tutor sequentially executes those scripts. We see two students interacting with the tutor object using text chat. The users activate the tutor by clicking on it (touch stimuli).

Connectivity between the tutor and SecondLife environment is enabled using a HTTP Middleware [31]. The component network of the SecondLife tutor, shown graphically in Fig. 6, is made of nine components and twelve connections. It receives two types of stimuli from the SecondLife environment, i.e.,

1. the user touching the agent to activate it and
2. the user sending a message in the agent's vicinity

When the tutor is activated, a *LaunchEvent* is propagated to the *GreetingActor* and the *TutoringManager*. *GreetingActor* sends a greeting message back to the environment via the *OutputCoordinator* and the *SLActor*. The *TutoringManager* encapsulates the TuTalk dialog engine.

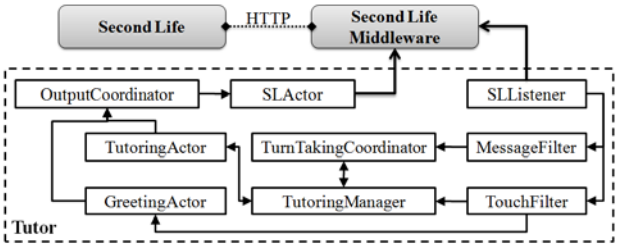


Fig. 6. Component Network of the Second Life Tutor

When triggered, it starts tutoring by sequentially executing the available TuTalk scripts. Tutor turns (questions and feedback) are sent to the environment via the *TutoringActor*, the *OutputCoordinator* and the *SLActor*. Student answers received via the *MessageFilter* are collected by the *TurnTakingCoordinator* when the Tutor is expecting the students to respond to its question (e.g. after the first turn of Table 2). If multiple students respond within a few moments of each other to the same tutor turn, the *TurnTakingCoordinator* accumulates those responses and sends them to the *TutoringManager* to be matched to the tutor's question. The *TutoringManager* filters responses collected by the *TurnTakingCoordinator* through prioritization rules that look for a correct or an incorrect answer, i.e., a relevant response to the tutor's questions.

Here we have adopted a solution that is specific to the situation of tutoring interaction to deal with the problem of not knowing the addressee of a student turn since responses that are relevant to the tutor's question can be considered as addressed to the Tutor. If no relevant responses are received for an extended amount of time, the *TutoringManager* can employ interaction tactics such repeating the question, urging the students to respond, giving a hint or moving on, depending on various conditions.

As mentioned earlier, the flexibility to incorporate components such as the *TurnTakingCoordinator* and *TutoringManager* is an example of how Basilica allows developers to address the complex interaction dynamics of multi-user interactive situations.

3.5 Agent Development

The process of developing agents using the Basilica architecture begins with design iterations that identify the required capabilities of the agent. Requirements are identified at this stage such as types of conversational displays by the agent (like tutoring, hinting, instructing, greeting, etc.), collaboration environments to be employed and use of specific input/output modalities. Next, developers create Basilica components that can satisfy these requirements. As these components are created, the events each component requires and produces are identified. As discussed in the next section, it is possible to re-use existing components to satisfy one or more of these requirements. At the third stage, the components are connected to each other through a network such that components that require certain events are connected to components that produce them.

Next, a generic agent factory provided by Basilica is configured to use the component network (specified as shown in Figure 4). At this stage, the agent can be instantiated and tested. It is often necessary to iterate through this development process several times before the required components, events and connections are identified, created and configured completely. The tools described ahead can be used to aid this agent development process.

TABLE 2
EXCERPT OF A CONVERSATION BETWEEN TWO STUDENTS AND THE SECOND LIFE TUTOR

Tutor	Consider the effect of T-max on the efficiency of a cycle. Do you know whether T-max should be kept high or low?
Student1	we should keep tmax high
Tutor	But let's think about one important consequence of keepint T-max high. At high T-max what happens to the quality of teams.
Student2	it increases i believe
Tutor	The turbine exit will have better steam quality when T-max is increased. So increasing steam quality and efficiency means high T-max is good. Nevertheless, there are limitations to how high you can increase T-max. There are mainly two problems. Can you name them?
Student2	safety is one... efficiency is the other?
Tutor	Mainly, safety concerns and exceeding the maximum temperature the material can withstand. Keeping Tmax high increases efficiency and also steam quality. Make sure you and your partner both understand these ideas.

Re-Use of Components. We have a growing set of behavioral components which can be re-used to build tutors for several learning situations. For example, as shown in the example in the previous section, the *TutoringManager* and the *TutoringActor* can be used to include tutoring scripts developed for the TuTalk system within the agent's interactive behavior. The agents discussed in the next section show extensive re-use of these components.

Furthermore, the interaction with the SecondLife environment is isolated to the *SLListener* and *SLActor* components. These components can be replaced to make the same agent work in other similar environments (like chatrooms or other multi-user virtual environments). It is useful to allow agents to operate in multiple environments with comparable affordances especially in the online learning situation to allow the students to be able to interact with the agent from an environment of their choice.

Overall, decomposing agents into small, loosely coupled components that encapsulate behavior allows application of object-oriented programming principles that facilitate incremental and distributed development in teams. Just as these principles have enabled large scale software development, we believe that they will facilitate development of complex and highly interactive instruc-

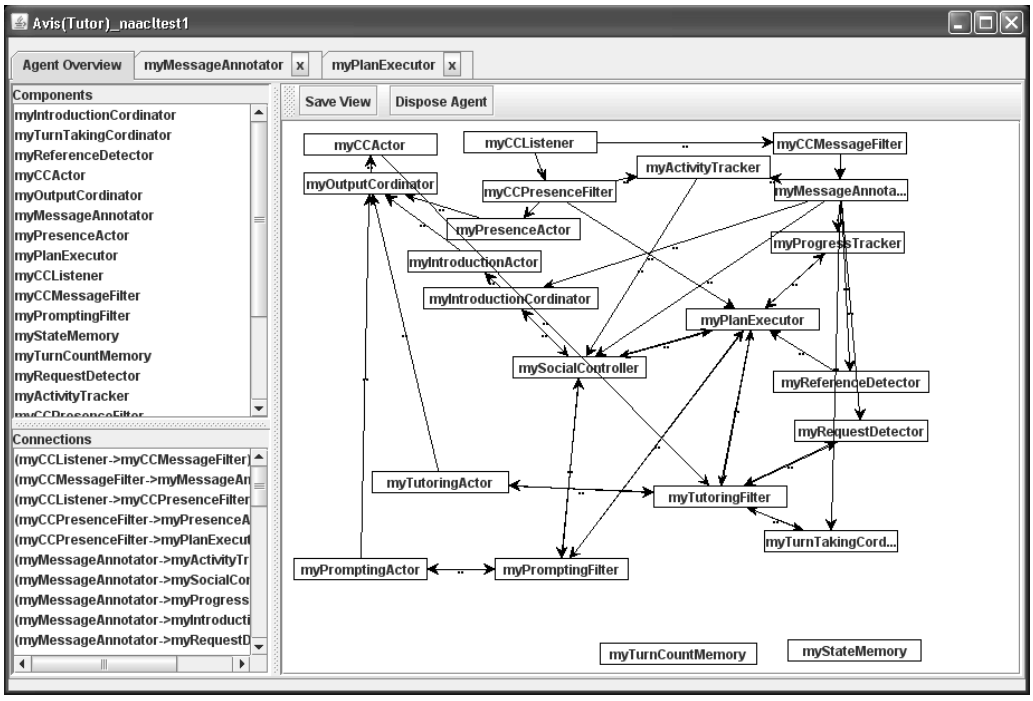


Fig. 7. Basilica Visual Debugging Interface

tional agents for mass use.

Development Tools. Besides the core classes of the architecture, Basilica provides a variety of debugging utilities through loggers and observer classes. A visual debugging interface is available as a part of these utilities to help developers verify the connections between components and track event propagation as developers proceed through the agent development iterations. Fig. 7 shows a screenshot of this debugging interface. The component network shown in the interface is animated as events propagate through the network. Developers can click on any component or connection to get a detailed look at the events generated and processed by each component.

To support the deployment of agents built using the Basilica architecture for large experiments, the architecture provides an operation class that can launch and manage several agents. Another utility built within the architecture is the Timer which can has been useful for implementing behaviors that require periodic involuntary stimuli rather than external stimuli from the environment (e.g., to regularly check for student participation).

3.6 Related Work

We can compare the Basilica architecture to other work on architectures for creating CAs as well as to architectures for supporting CSCL.

Various approaches [17] [20] and corresponding software architectures [32] have been proposed to create CAs. As discussed in section 2.1., these approaches employ a very high level language to represent the agent capabilities, which makes it difficult to create highly interactive behaviors that are not only useful but increasingly necessary when the agents are interacting with more than one student. Furthermore, these architectures are based on the assumptions discussed in section 2.2 that do not generalize to collaborative learning situations. For example, the Ravenclaw architecture [21] processes user utterances only when the agent is in the input phase, i.e., the agent is expecting an input. However, while supporting collaborative learning, agents must constantly monitor student interaction and perform interventions when necessary. Basilica allows this as demonstrated in section 4.2 by creating components dedicated to detect characteristics of student interaction (e.g. inactivity). Similarly, the AutoTutor system utilizes a five-step dialogue frame [32] that prohibits input from the student at specific steps. The software architecture underlying these systems can be extended to add additional desirable behaviors. In general such extensions would lead to implementation of an event-driven interaction model similar to the one suggested in Section 3.1.

On the other hand architectures proposed to support collaborative learning have largely focused on enabling collaboration between students by creating virtual environments that provide rich awareness, communication modalities, visualization of shared knowledge, integration of external resources and allow implementation of collaboration scripts [33]. Use of agents as adaptive support has been explored in limited form by supporting

specific learner roles like Peer tutors [25]. On the other hand, agents built using the Basilica architecture are capable of supporting the learning group as a whole by employing situation specific interactive behaviors such as turn taking as discussed in section 3.4. Similarly agents utilized in asynchronous environments [34] to support learners bypass the issues of interacting with more than one learner simultaneously because the students do not interact with each other in real time.

4 Conversational Agents for Collaborative Learning

The approach of building CAs using the Basilica architecture is applicable to a variety of agents that are situated in complex, extended, multi-party interactive situations. In the rest of the paper, we will focus on our work on developing interactive tutors that support teams of two or more students in multiple educational domains. Furthermore, we will show that these agents have been developed for a variety of collaborative environments as appropriate for the specific learning situation. We will discuss the implementations of three agents developed by us using the Basilica architecture to highlight how these agents showcase the strengths of the proposed architecture. It may be noted that extended discussion on evaluation of the agents developed using this architecture is beyond the scope of this article. A summary of these evaluations is provided in section 4.5.

4.1 CycleTalk Tutor

CycleTalk is an intelligent tutoring system that helps sophomore engineering students learn principles of thermodynamic cycles (specifically Rankine Cycle) in the context of a power plant design task. Teams of two students work on designing a Rankine cycle using a Thermodynamics simulation software package called CyclePad [32]. As part of the design lab during which this learning task is performed, students participate in collaborative design interaction for 30-45 minutes using ConcertChat, a text based collaboration environment [36] shown in Fig. 8. Our earlier work [1] has shown the pedagogical benefit of this collaborative design activity.

An automated tutor participates in the design interaction along with the two students. The CycleTalk tutor provides instructional support to the students to ensure that they learn the underlying thermodynamic concepts as they design. The CycleTalk tutor was the first tutor implemented using the Basilica architecture and has been modified over the last three years in accordance with the evolution of the research studies conducted using this tutor. Improvements to the CycleTalk tutor served as requirements for improving the Basilica architecture. Specifically, these improvements included development of various types of components, efficiencies in the architecture's event propagation mechanism and creation of suitable abstractions and interfaces to the architecture's core classes to facilitate development and re-use. Some of these aspects are discussed ahead as we present a recent implementation of the CycleTalk tutor implemented using the Basilica architecture.

While we have shown that instructional support provided by automated tutors is effective, as compared to students working individually, we have observed that teams of students often ignore and abuse the tutor [1]. In our recent studies in this Thermodynamics learning domain, we have investigated the use of interaction strategies that can help in engaging the students more deeply in the instructional conversation with the tutors. One of these strategies (*Attention Grabbing*) was designed to in-

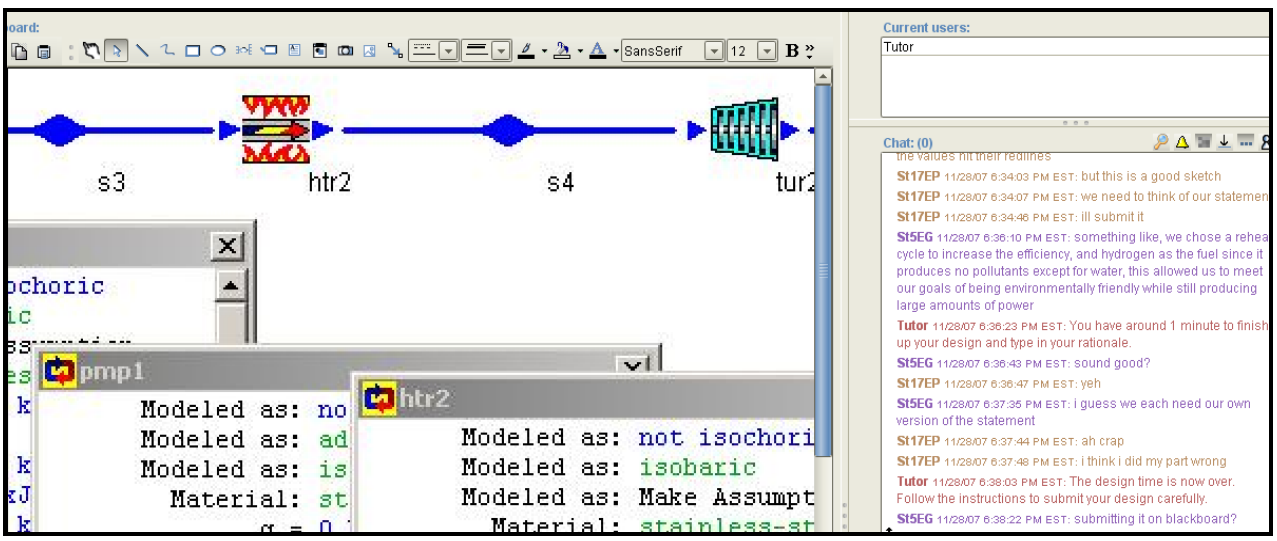


Fig. 8. ConcertChat Collaboration Environment

TABLE 3
EXCERPT SHOWING THE ATTENTION GRABBING STRATEGY (TURN 61)

59	09:35:07	**Student2**	the pressure stays constant through the boiler?
60	09:35:47	**Student1**	Yeah
61	09:35:55	**Tutor**	Now might be a good time for some reflection.
62	09:36:00	**Tutor**	Consider the effect of increasing Qin. What happens to power out when Qin is increased?
64	09:36:46	**Student2**	i am pretty sure we want high Qin

trusively grab the student's attention [3]. It prompts the students to pay attention to the tutor before the tutor starts the instructional conversation. An excerpt of this strategy is shown in Table 3. The tutor prompts the students (Turn 61) to grab their attention and waits for a silence (5 seconds) among the students to infer that the students are now paying attention to the tutor.

Another strategy (*Ask when Ready*) developed as an improvement to the *Attention Grabbing* strategy informed the students that the tutor has a relevant instructional topic to discuss and asks them to let the tutor know when they were ready to talk about the topic [4]. This strategy allows the students to complete their current topic of discussion before engaging in conversation with the tutor. An example of this strategy is shown in Table 4. The tutor informs the students that it is ready to talk about P-max (maximum pressure in a Rankine cycle) in turn 41 & 42. The students finish their current topic of discussion and indicate that they are ready to discuss P-max in turn 47. Note that the turn 41 is similar in principle to the attention grabbing prompt shown in turn 61 in Table 3.

Both these agents employ the turn-taking and tutoring components discussed in the example agent in the previous section. Fig. 9 shows the component network implemented for a tutor that employs the *Ask when Ready*

TABLE 4
EXCERPT SHOWING THE ASK WHEN READY STRATEGY (TURN 41 & 42)

40	08.52.24	**Student5**	and then Power out vs. the same things
41	08.52.26	**Tutor**	Lets review the effect of changing P-max on the cycle.
42	08.52.27	**Tutor**	Type: HELP WITH PMAXKCD if you want to discuss it with me.
...			...
47	08.54.08	**Student7**	HELP WITH PMAXKCD
48	08.54.14	**Tutor**	When P-max increases, is the need to reject heat from the cycle increased or decreased?
49	08.54.51	**Student5**	decreased

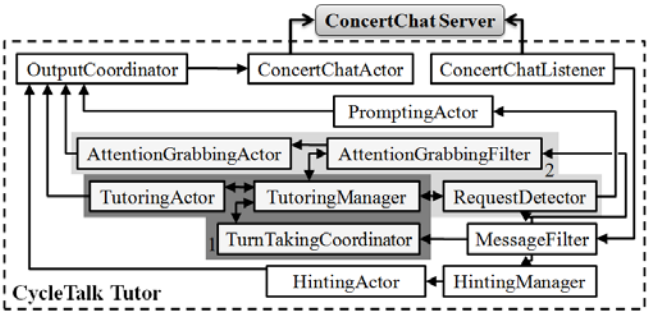


Fig. 9. Component Network of the CycleTalk Tutor

strategy. It is made of 13 components and 21 connections. There are six types of components, i.e., *Listeners*, *Actors*, *Filters*, *Detectors*, *Coordinators* and *Managers*. *Listeners* listen to stimuli from the environment and translate them into events internal to the agent. *Actors* perform actions which may be directly observable by other participants in the environment. *Filters* process information that events carry and propagate them further based on their programmed conditions. *Detectors* are special kinds of Filter which detect specific semantic concepts/phrases and send out a detection event. *Coordinators* control the flow of events between related components to achieve coordinated behavior. *Manager* components exhibit a variety of behavior like planning, execution and control. As we discuss other agents built on this architecture, we will discover other types of components being used.

We can note that components shown in the shaded area labeled as 1 in Fig. 9 are connected the same way as those components in the example agent. In order to implement the behavioral capabilities that allow the agent to use the *Attention Grabbing* and *Ask when Ready* interaction strategies, we have added three new components shown in the shaded area labeled 2. When the *TutoringManager* decides that an instructional topic is relevant to the current discussion, it informs the *AttentionGrabbingFilter* to grab the students' attention using an appropriate prompt. The *TutoringManager* also informs the *RequestDetector* to look out for the appropriate trigger phrase. Once the trigger phrase is detected, the *TutoringManager* starts the TuTalk script corresponding to the requested instructional topic. Besides the tutoring behavior, the CycleTalk tutor has a hinting behavior implemented using the *HintingManager* and *HintingActor* that use a topic model to provide relevant hints based on the interaction between the students [1].

Using the Basilica architecture to develop the CycleTalk tutor has supported this line of investigation in multiple ways. Foremost, it may be noted that components corresponding to behaviors that do not change due to our experimental manipulations can be isolated within the agent network. This allows us to incrementally add beha-

vioral components that implement these interventions. Further, since we use a programmatic approach to building these agents, we are not restricted to a small set of operators provided by typical agent authoring languages making it possible to implement strategies like *Attention Grabbing* and *Ask when Ready*. Finally, the ability to integrate existing natural language processing modules as *Filter* components makes Basilica a helpful architecture for creating complex and highly interactive conversational agents.

Besides implementing strategies for initiating tutoring conversations with a team of students, we have been investigating the use of role assignment to the students [3] [5]. Students in each team are divided into Pro-Environment and Pro-Power roles to elicit broad coverage of arguments within the team during the design interaction. In order to support both the students while keeping track of their different roles, a recent implementation of the CycleTalk tutor agent [5] employs a *GoalManager* component. This component identifies student roles based on their design goals i.e., whether they are Pro-Power or Pro-Environment. The agent then reinforces the need to explore both goals by choosing different version of tutoring scripts to emphasize the two different goals.

4.2 WrenchTalk Tutor

Freshmen Mechanical engineering students participate in a Wrench design lab as they learn concepts of force, moment and stress. We have developed an agent that supports teams of three to five students learning these underlying concepts as they work on the wrench design task. The students are provided with a basic wrench specification and asked to design a better wrench in terms of safety, ease of use and material costs. As a team, the students discuss and decide new values of dimensions and materials for the wrench they design. Students use the ConcertChat environment to interact with each other and the agent. Besides the text-based chat, the whiteboard provides a way for the team to share their designs and calculations. Like the CycleTalk tutor, the WrenchTalk tutor provides information about the learning task and delivers instructional content at appropriate time in the form of hints and TuTalk scripts.

A requirement that some of the agents developed using the Basilica architecture posed included the need for a way to share internal state with other components. A generic memory component was developed to serve this purpose. Since the behavior of the memory components involves only *commit* and *indexed retrieval,* which are quite fast computationally, the Memory component utilizes a synchronous event processing mechanism unlike other components. The WrenchTalk tutor was among the first agents to use this type of a component. Another design pattern that was explored in this agent was the use of two manager components that operate in tandem with each other by sharing control as discussed ahead.

TABLE 5
SOCIAL INTERACTION STRATEGIES

1. Showing Solidarity
1a. Do Introductions
1b. Be Protective & Nurturing
1c. Give Re-assurance
1d. Complement / Praise
1e. Encourage
1f. Conclude Socially
2. Showing Tension Release
2a. Expression of feeling better
2b. Be cheerful
2c. Express enthusiasm, elation, satisfaction
3. Agreeing
3a. Show attention
3b. Show comprehension / approval

One of the experimental interaction strategies explored in the Wrench design lab is motivated from research in small group communication. Interaction process analysis on small group interaction [37] has enumerated twelve interaction categories. Six of these categories are related to task related interaction whereas the other six are related to socio-emotional interaction. Research on conversational agents has largely ignored non-task aspects of communication. As we move towards use of conversational agents in multi-party situations, we believe that agents must be capable of performing both task as well as socio-emotional interaction so that they can participate in all aspects of group interaction. Based on the three positive socio-emotional interaction categories enumerated by Bales, we have developed eleven social interaction strategies listed in Table 5.

Table 6 shows an excerpt of an interaction that shows the tutor's use of some of these interaction strategies. Turn 142 concludes a TuTalk script about the relationship of stress and ease of use of a wrench. At turn 144, the tutor complements (Strategy 1d) one of the students for participating in the discussion. Also note that one of the students exhibits enthusiasm about their team's designs in turns 143 and 146. The tutor exhibits cheerfulness (Strategy 2b) in turn 145 and 147 to reciprocate the student's enthusiasm.

An experimented conducted in November 2009 [6] involved the evaluation of these social interaction strategies. We implemented a tutor using the Basilica architecture, which is capable of performing these social behaviors alongside the tutoring behavior. Fig. 10 shows the

TABLE 6
EXCERPT OF A CONVERSATION BETWEEN THREE STUDENTS AND WRENCHTALK TUTOR

139	**Tutor**	One last thing on this topic, Does more (or less) stress in a wrench make it easier to use?
140	**S95**	no change?
141	**Tutor**	You are correct about that.. Stress doesn't determine ease of use.
142	**Tutor**	It's the moment achieved by the wrench that determines the ease of use.
143	**S89**	yay!
144	**Tutor**	It's good to have your contributions in the discussion Jackie :-)
145	**Tutor**	Go team :-)
146	**S89**	Go team yay
147	**Tutor**	I am happy to be working with our team
148	**S89**	Me too
149	**S95**	whoa the bot knows my name
150	**Tutor**	Unfortunately maximum stress (12800) in our design1 is way above the maximum allowed stress (i.e. 8750)
151	**Tutor**	This wrench cannot be safely used!

components network of the WrenchTalk tutor. We use two specialized types of manager components (*PlanExecutor* and *SocialController*) in this network. The *PlanExecutor* is responsible for executing the tutor's task-related interaction plan, which is comprised of 37 steps. The plan is executed largely sequentially; however the plan controller can choose to skip some steps in the interest of time. The *SocialController* implements the eleven social interaction strategies listed earlier. The strategies are triggered by rules based on combinations of four features: the most recent plan step, semantic annotations of the most recent student turns (by the *AnnotationFilter*), activity levels of each student (measured by the *ActivityDetector*) and the percentage of tutor turns generated by *SocialController* to

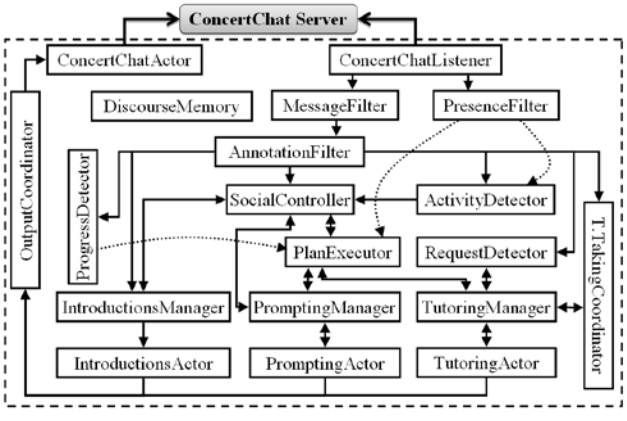


Fig. 10. Component Network of the WrenchTalk Tutor

regulate the amount of social behavior by the tutor. Once the controllers determine a step or a strategy that is to be generated they are conveyed to the corresponding observable behavior generation components (*PromptingManager*, *TutoringManager*, *IntroductionsManager*).

The flexibility that the Basilica architecture provides is demonstrated in this case by the use of the two primary controllers (*PlanExecutor* and *SocialController*), unlike typical dialog systems, which use only one primary controller (Dialog Manager). Coordination between the two controllers is achieved by connecting them. For instance, when the *PlanExecutor* is working, it blocks the *SocialController* and vice versa. Control is shared between the two by transferring control at the end of every interaction step. We also notice the re-use of the tutoring components from the agents discussed earlier.

The *ActivityDetector* component provides a way to monitor the uneven student participation by students which is characteristic of multi-party interaction. When one or more students become critically inactive in the interaction, this component can choose to trigger Strategy 1e listed in Table 5 through the *SocialController* to encourage the inactive students to contribute to the learning activity.

An additional type of component we see in this network is the memory component (*DiscourseMemory*). It maintains a history of the discourse state messages and interaction steps. Memory components circumvent the usual event propagation mechanism by using a specialized faster commit / retrieve mode of access from other components. This is to reduce the computational overhead of using two events (request and response) otherwise required for accessing memory components.

An experiment we conducted to evaluate of the WrenchTalk tutor also demonstrates the usefulness another feature of the Basilica architecture, i.e., Observer components. In order to compare our implementation of the social interaction strategies with human quality social interaction, we augmented the tutor with a human observer user interface that allowed a human tutor to insert social prompts at any time during the interaction with the students. The students would see that human modified prompt as another prompt from the tutor. This was implemented by assigning an observer to the *AnnotationFilter* and *PlanExecutor* components.

4.3 PsychChallenge Peer

The Basilica architecture does not make any specific assumptions about the role of the agent in the interaction. Agents supporting learners may not only be tutors. In this section we will discusses an agent that demonstrates the ability to build agents that other roles besides a tutor.

PsychChallenge is a vocabulary game that is part of a

learning portal of an introductory psychology text book. Students play this game as part of an assignment as they progress through each chapter of the book. The game involves collaboratively learning vocabulary related to each chapter by helping each other guess domain vocabulary terms by providing hints about the terms. One of the players takes on the role of a hint *Giver* and the other players play the *Guesser* role. The game is accessible to the students through a special purpose web-interface.

Students can choose to play the game with other students who are online at the same time. Teams (or individuals) can choose to add the PsychChallenge peer agent to the game. Note that this agent is different from the other agents discussed in this paper in that it plays the role of a peer with respect to the student instead of a tutor. Agents used in this role are comparable to the use of agents as learning companions [38].

Fig. 11 shows the component network of the peer agent. The agent is connected to the web-interface through a middleware component that operates in a way similar to the SecondLife HTTP middleware [31]. The peer agent plays the role of the *Guesser* or the *Giver* based on the role that it is assigned by the game. The *GuessingActor* and the *HintingActor* components are augmented with instructional content corresponding to each term in the game's vocabulary to allow the agent to behave intelligently.

When the agent is playing the *Guesser* role it tries to elicit better hints from the *Giver*. When the agent is the *Giver*, it provides useful hints to the students to help them guess the correct term. Unlike the role modeling discussed in section 4.1 where the agent displays instructional content based on its awareness of student roles; this peer agent employs additional interaction tactics especially when it is the *guesser* to encourage other students who might also be *guessers* to participate in the learning game. Other behavior that this Peer agent displays includes greeting the students at the start of the game and informing them about its role at the change of every round.

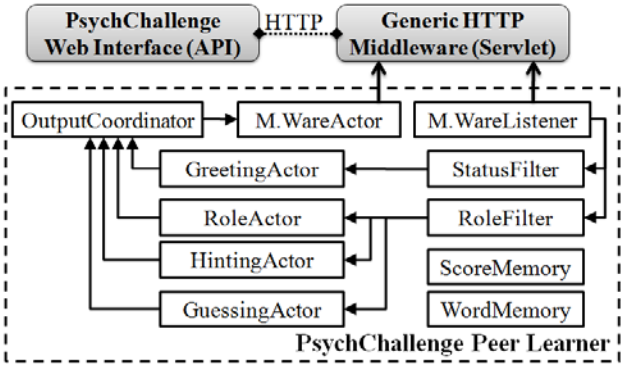


Fig. 11. Component Network of the PsychChallenge Peer

4.4 Types of Behavioral Components

Based on the three agents developed using the Basilica architecture that are described in section 4.1-4.3 we can identify the characteristics of different types of behavioral components that are necessary and/or common to most agents implemented using Basilica. Some of these types were listed in section 4.1.

Foremost of these are the environment listener and environment actor components. They are necessary for all agents as they integrate the agent with a collaboration environment such as SecondLife (*SLListener* and *SLActor*) or ConcertChat (*ConcertChatListener* and *ConcertChatActor*). These can be readily re-used for new agents being developed for already supported environments. On the other hand, these components are among the first components to be implemented while developing agents for a new environment.

We can find multiple filter and detector components being used among the three agents. They perform a variety of operations (like parsing, classification, annotation, etc.) and transformations on the data encapsulated within events they receive. Further these components can be used to control the flow of events. For example the *MessageFilter* keep the presence events from the environment from propagating to components that do not need those events. All agents the process student input have atleast one of these components.

Different types of manager components make up the controllers of the different user observable behaviors of the agent. These managers keep track of knowledge resources and interaction states for the behaviors they perform and provide triggers to other components that realize the behaviors these managers control. All agents that actively interact with students have atleast one of these manager components. Note that for very simple behaviors the management logic is sometimes built within the actor components that realize those behaviors (e.g. the *GreetingActor* in Fig. 6 and Actors in Fig. 11).

Memory components while not mandatory to any agent implementation are often used for agents that need to have several managers. Finally, we find that all agents implement several special purpose components like actors that realize the user-observable behaviors and coordinators that facilitate agent participation in complex interaction dynamics. As discussed in section 3.5, many of these components can be reused among agents that display the same behaviors.

4.5 Summary of Evaluations

We have conducted a number of experiments in undergraduate mechanical engineering courses using the multiple versions of the CycleTalk Tutor and the WrenchTalk Tutor described earlier. These experiments evaluate the

effectiveness of the support these agents offer. In this section, we present a summary of the results from these evaluations.

In an early experiment involving the use of our CycleTalk tutor agent [1], students worked in pairs on a power plant design task. We found that the student pairs which were supported by our tutor agent learned 1.24σ more than control condition students who worked on the same learning task without a partner or a conversational agent. Students who worked either with a partner or with a conversational agent learned 0.9σ and 1.06σ respectively compared to students in the control condition.

In subsequent studies [3, 4] with the CycleTalk tutor agent, we found that use of the *Attention Grabbing* strategy described in section 4.1 significantly increased the number of relevant responses from the students to the tutor's instructional turns. While this did not translate into significant learning gains, we observed the use of such interactive strategies was the key to improving the quality of interaction with students and tutors in collaborative learning settings. Improving the *Attention Grabbing* strategy to the *Ask when Ready* strategy [4] led us to a learning gain of 0.8σ over our previous state of the art tutors [1].

On a parallel track investigating the use of social displays by agents in collaborative learning settings, we found that use of small talk to engage middle school students before presenting a word problem [2] to them led to a marginal learning effect of 0.55σ. The students who received the experimental intervention (small talk) also perceived their partners and themselves to be more helpful to each other. In another experiment, the tutor's use of motivational prompts during extended design interactions led to significant improvements in the attitude of the students towards the tutor. Most recently, we found that the agents that use systematically designed social interaction strategies [6] listed in Table 5 achieved a significant learning effect of 0.71σ compared to our earlier state of the art agents [4]. Also, based on results from a perception survey, we find students prefer the agents with enhanced social capabilities.

Basilica has enabled the implementation of these new interactive behaviors as demonstrated in the examples discussed earlier. In general we find the design and implementation of appropriate interactive behaviors is the key to improving the state of the art in supporting collaborative learning through the use of conversational agents.

5 Conclusion & Directions

We have presented the Basilica architecture, which adopts a programmatic approach to develop Conversational Agents that are composed of a network of behavioral components. Through a discussion of three interactive agents developed using this architecture, we have demonstrated the capabilities of this architecture at addressing the three requirements that motivated the design of this architecture. Specifically,

1. Basilica provides a rich representational capability unlike other agent development frameworks by not restricting agent behavior to a small set of communicative acts/operators.
2. A component network can be built using Basilica to address situation-specific complex interaction dynamics encountered in extended multi-party interactive situations like collaborative learning.
3. Adoption of objected-oriented programming principles and availability of a variety of support tools along with the architecture helps in making the development process systematic, distributable and efficient.

While most of the agents developed using this architecture support collaborative learning as tutors, in Section 4.3 we also discussed an agent that plays the role of another student to enable collaborative learning when only one student is available. It can also be used as an additional student in a group of human students. Furthermore, we have used this architecture to support student pairs who have been assigned different task roles that demand the tutor to be able to identify the roles and display instructional behaviors that support the different roles of the students.

Moving forward with the use of this architecture for building interactive agents for educational purposes, we plan to develop a mechanism that allows multiple agents to participate in an interaction (e.g., two tutors, or a tutor and bunch of automated students), with a team of students. Another desirable future capability of the architecture is with regard to being able to scale up the production and deployment of such agents. Deployment of agent operations that served thousands of students would require an ability to distribute the agent's computation over several machines. In principle, since sections of the component network are loosely coupled, such distribution is imminently practical with the Basilica architecture. Finally, a mechanism for discovery and sharing of behavioral components across different research groups using the architecture needs to be facilitated through a component repository. Development of an ontological specification schema to classify components and events would facilitate organization and retrieval of these components.

While Basilica provides us with a way to create agents that participate in collaborative learning interactions, it only enables the exploration of an issue that is central to the design and development of effective conversational

agents i.e., the issue of identifying and creating instructional, social and other interactive behaviors that the agents must display to stand their ground and achieve their pedagogical objectives. Through the three agents described in this paper, we can notice that some of these behaviors are re-useable while others are specific to the interactive situation.

In our work, we have used this architecture to implement and evaluate the effectiveness of a variety of interactive behaviors such as the *Attention Grabbing*, *Ask when Ready* and *Social Interaction Strategies* along with other state of the art pedagogical strategies like feedback and hinting. A continued exploration of these behaviors/strategies must be pursued in order to improve the state of art in interactive support for collaborative learning.

In summary, we find that the proposed architecture is a suitable platform for research in interactive support for collaborative learning and offers a potential solution for mass-development and deployment of such systems.

ACKNOWLEDGEMENT

This work was supported in part by the following grants: NSF SBE 0836012, NSF DRC 0835426, NSF EEC 0935145.

REFERENCES

[1] R. Kumar, C. P. Rosé, M. Joshi, Y. C. Wang, Y.Cui and A. Robinson, "Tutorial Dialogue as Adaptive Collaborative Learning Support", *Proc. Artificial Intelligence in Education (AIED)*, 2007

[2] R. Kumar, G. Gweon, M. Joshi, Y. Cui, and C. P. Rosé, "Supporting students working together on Math with Social Dialogue", *Proc. Workshop on Speech and Language Technology in Education (SLATE)*, 2007

[3] S. Chaudhuri, R. Kumar, M. Joshi, E. Terrell, F. Higgs, V. Aleven, and C. P. Rosé, "It's Not Easy Being Green: Supporting Collaborative Green Design Learning", *Proc. Intelligent Tutoring Systems (ITS)*, 2008

[4] S. Chaudhuri, R. Kumar, I. Howley, and C. P. Rosé, "Engaging Collaborative Learners with Helping Agents", *Proc. Artificial Intelligence in Education (AIED)*, 2009

[5] H. Ai, R. Kumar, D. Nguyen, A. Nagasunder, and C. P. Rosé, "Exploring the Effectiveness of Social Capabilities and Goal Alignment in Computer Supported Collaborative Learning", *Proc. Intl. Conf. on Intelligent Tutoring Systems (ITS)*, 2010

[6] R. Kumar, H. Ai, J. L. Beuth, and C. P. Rosé, "Socially-capable Conversational Tutors can be Effective in Collaborative-Learning situations", *Proc. Intl. Conf. on Intelligent Tutoring Systems (ITS)*, 2010

[7] E. Arnott, P. Hastings, and D. Allbritton, "Research Methods Tutor: Evaluation of a dialogue-based tutoring system in the classroom", *Behavior Research Methods*, vol. 40, no. 3, pp. 694-698, 2008

[8] A. C. Graesser, P. Chipman, B. C. Haynes, and A. Olney, "AutoTutor: An Intelligent Tutoring System with Mixed-initiative Dialogue", *IEEE Transactions in Education*, vol. 48, pp. 612-618, 2005

[9] A. Renkl, R. Stark, H. Gruber, and H. Mandl, "Learning from worked-out examples: The effects of example variability and elicited self-explanations", *Contemporary Educational Psychology*, Vol. 23, pp 90-108, 1998

[10] V. Aleven and K. R. Koedinger, "The need for tutorial dialog to support self-explanation", C. P. Rose and R. Freedman (Eds.) *Building Dialogue Systems for Tutorial Applications, Papers of the 2000 AAAI Fall Symposium,* pp. 65-73, Menlo Park, California: AAAI Press, 2000

[11] I. Kollar, F. Fischer, and F. W. Hesse, "Computer-supported cooperation scripts - a conceptual analysis", *Educational Psychology Review*, 2006

[12] A. Weinberger, "Scripts for Computer-Supported Collaborative Learning Effects of social and epistemic cooperation scripts on collaborative knowledge construction", Unpublished Dissertation, University of Munich, 2003

[13] M. Baker, and K. Lund, "Promoting reflective interactions in a CSCL environment", *Journal of Computer Assisted Learning,* vol. 13, 175-193, 1997

[14] A. M. O'Donnell, "Structuring Dyadic Interaction through Scripted Cooperation", In O'Donnell & King (Eds.), *Cognitive Perspectives on Peer Learning*, New Jersey: Lawrence Erlbaum Associates, 1999.

[15] K. Stegmann, A. Weinberger, F. Fischer, and H. Mandl, "Scripting Argumentation in computer-supported learning environments", In P. Gerjets, P. A. Kirschner, J. Elen & R. Joiner (Eds.), *Instructional design for effective and enjoyable computer- supported learning. Proc. Joint meeting of the EARLI SIGs Instructional Design and Learning and Instruction with Computer,* pp. 320-330, 2004.

[16] C. P. Rosé, Y. C. Wang, Y. Cui, J. Arguello, K. Stegmann, A. Weinberger, and F. Fischer, "Analyzing Collaborative Learning Processes: Automatically Exploiting the Advances of Computational Linguistics in Computer-Supported Collaborative Learning", *International Journal of Computer Supported Collaborative Learning*, vol. 3, no. 3, pp 237-271, 2008

[17] C. P. Rosé, D. Bhembe, S. Siler, R. Srivastava, and K. VanLehn, "Exploring the Effectiveness of Knowledge Construction Dialogues", *Proc. Artificial Intelligence in Education (AIED)*, 2003

[18] VoiceXML, http://www.w3.org/TR/voicexml21/, 2007

[19] P. Jordan, B. Hall, M. Ringenberg, Y. Cui, and C. P. Rosé, "Tools for Authoring a Dialogue Agent that Participates in Learning Studies", *Proc. Artificial Intelligence in Education (AIED)*, 2007

[20] R. Freedman, "Plan-based dialogue management in a physics tutor", *Proc. of 6th Applied Natural Language Processing Conference*, pp 52-59, 2000

[21] D. Bohus, A. I. Rudnicky, "The RavenClaw dialog management framework: Architecture and systems", *Computer Speech & Language*, vol. 23(3), pp. 332 - 361

[22] A. Raux, "Flexible turn-taking in Spoken Dialog Systems", PhD Dissertation, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, 2008

[23] Š. Benuš, "Are we 'in sync': Turn-taking in collaborative dialogues", *Proc. Interspeech*, 2009

[24] D. Bohus, "Error Awareness and Recovery in Conversational Spoken Language Interfaces", PhD dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 2007

[25] D. Diziol, E. Walker, N. Rummel, and K. R. Koedinger, "Using Intelligent Tutor Technology to Implement Adaptive Support for Student Collaboration", *Educational Psychology Review*, vol.22(1), pp.89-102, 2010

[26] K. Kreijns, P. A. Kirschner, and W. Jochems, "Identifying the

pitfalls for social interaction in computer-supported collaborative learning environments: a review of the research", *Computers in Human Behavior*, vol.19, pp. 335-353, 2003
[27] G. Gweon, S. Jeon, J. Lee, and C. P. Rosé, "Diagnosing Problems in Student Project Groups", In S. Puntambekar, G. Erkens, C. Hmelo-Silver, C. (Eds.), *Analyzing Collaborative Interactions in CSCL: Methods, Approaches and Issues*, Springer, In press
[28] Stefan Larson, "Dialogue Systems and Projects", http://www.ling.gu.se/~sl/dialogue_links.html, 2006
[29] I. O'Neill, P. Hanna, X. Liu, and M. McTear, "The Queen's Communicator: A Object-Oriented Dialogue Manager", *Proc. Eurospeech*, pp 593-596, 2003
[30] P. Jordan, M. Rigenberg, and B. Hall, "Rapidly developing Dialogue systems that support learning studies", *Proc. ITS Workshop on Teaching with Robots, Agents and NLP*, 2006
[31] B. K. A. Weusijana, R. Kumar and C. P. Rose, "MultiTalker: Building Conversational Agents in Second Life using Basilica", *Second Life Education Community Convention, Purple Strand: Educational Tools and Products*, 2008
[32] A. C. Graesser, M. Jeon, and D. David, "Agent Technologies Designed to Facilitate Interactive Knowledge Construction", *Discourse Processes*, vol. 45:4, pp.298-322, 2008
[33] F. Loll, N. Pinkwart, O. Scheuer, and B. M. McLaren, "Towards a Flexible Intelligent Tutoring System for Argumentation", *Proc. of 9th IEEE Intl. Conf. on Advanced Learning Technologies (ICALT)*, Los Alamitos, CA, 2009
[34] W. Chen, J. Dolonen, and B. Wasson, "Supporting Collaborative Knowledge Building with Intelligent Agents", In: V. Palade, R, J, Howlett, and L. C. Jain (Eds.), *Knowledge-Based Intelligent Information and Engineering Systems, Advances in Artificial Intelligence. LNAI*, vol. 2774, pp. 238-244, 2003
[35] K. D. Forbus, P. B. Whalley, J. O. Evrett, L. Ureel, M. Brokowski, J. Baher, and S. E. Kuehne, "CyclePad: An Articulate Virtual Laboratory for Engineering Thermodynamics", *Artificial Intelligence*, vol. 114 (1-2), pp. 297-347, 1999
[36] M. Mühlpfordt, and M. Wessner, "Explicit referencing in chat supports collaborative learning", *Proc. Computer Support for Collaborative Learning (CSCL),* 2005
[37] R. F. Bales, *Interaction process analysis: A method for the study of small groups*, Cambridge, Massachusetts: Addison-Wesley, 1950
[38] C.-Y. Chou, T.-W. Chan, and C.-J. Lin, "Redefining the learning companion: the past, present, and future of educational agents", Computers & Education, vol.40, pp.255–269, 2003

Rohit Kumar is a PhD student at the Language Technologies Institute at Carnegie Mellon University. He received his Masters in Language Technologies at Carnegie Mellon University in 2007 and Bachelor in Engineering from Punjab Engineering College, Chandigarh in 2003. Rohit was a Research Scientist at Language Technologies Research Center, IIIT Hyderabad from 2003 to 2005. His research interests include Conversational Agents and their applications to variety of complex interactive situations. He is a member of IEEE and IEEE Computer Society since 2000.

Carolyn P. Rosé is an Assistant Professor with a joint appointment between the Language Technologies Institute and the Human-Computer Interaction Institute at Carnegie Mellon University. She completed her Master's degree in Computational Linguistics from Carnegie Mellon University in 1994 and her PhD in Language and Information Technologies also from Carnegie Mellon University in 1998. She has over 100 peer reviewed publications, mainly in top tier conferences and journals, spanning the fields of language technologies, learning sciences, educational technology, and human-computer interaction. She serves as the Secretary/Treasurer of the International Society of the Learning Sciences.