Submitted to 7th Int. Conf. on Advances in Computer-Human Interactions, Barcelona, Spain, March 2014.

A Real-Time Architecture for Embodied Conversational Agents: Beyond Turn-Taking

Bahador Nooraei,* Charles Rich and Candace L. Sidner
Worcester Polytechnic Institute
Worcester, MA, USA 01609
Email: rich|sidner@wpi.edu

***Abstract*—We describe the design, implementation and use of a middleware system, called DiscoRT, to support the development of virtual and robotic conversational agents. The use cases for this system include handling conversational and event-based interruptions, and supporting engagement maintenance behaviors, such as turn-taking, backchanneling, directed gaze and face tracking. The multi-threaded architecture of the system includes both "hard" and "soft" real-time scheduling and integrates with an existing collaborative discourse manager, called Disco. We have used the system to build a substantial conversational agent that is about to undergo long-term field studies.**

I. Introduction

Embodied conversational agents, both virtual and robotic, are becoming increasingly common and sophisticated in their interaction with humans. Originally, the transition from existing conversational agents (which typically interacted with users via alterating text input and output) to the "embodied" versions focused on animating a virtual face and/or body with expressions and gestures to complement the agent's utterances. More recently, however, conversational agents have acquired sensors, such as microphones, webcams and Kinects™, which are used to recognize users' speech and track their faces and bodies. Furthermore, many conversational agents now have *real* bodies, in the form of robots [1].

Along with these developments have come increasing expectations for more "natural" real-time interactions between agents and humans than the simple turn-taking of the earliest conversational agents. Unfortunately, the computational architectures underlying conversational agents have not evolved to keep up with the addition of sensors and bodies. Most such agents, including several that we have built, have simply added an ad-hoc collection of real-time mechanisms to what is basically the "letter" model of interaction discussed below. The goal of this work has been to make a fresh start and design a principled architecture that supports the continuous mutual signaling model discussed below.

Figure 1(a) shows the "letter" model of interaction, which still underlies most conversational agent implementations. In this model, one participant constructs an utterance (letter) containing the information to be communicated, sends it to the other participant, and waits. When the utterance is received, the other participant interprets it, thinks about the content, reacts (perhaps by performing an action) and then constructs a reply, if any, sends it back and waits. The cycle then repeats.

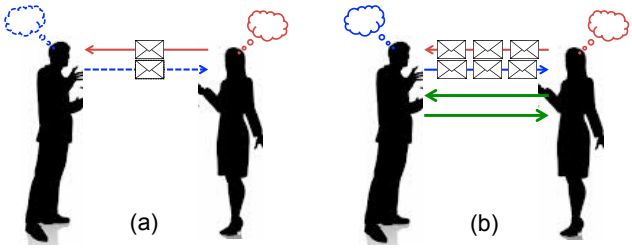

Figure 1. (a) Letter versus (b) continuous mutual signaling model.

This is a very poor model of how co-present humans naturally interact. In reality, human conversation is a kind of "continuous mutual signaling," as shown in Figure 1(b). In this model, both participants are continuously and simultaneously producing, observing and responding to behaviors, both verbal and nonverbal. Furthermore, the exact timing (synchronization) of these behaviors is a crucial part of the interaction.[1]

Thus communication in real conversations is multi-channel (multi-modal). The phenomenon of verbal turn-taking is just a strategy for managing the signal interference characteristics of one particular communication channel, namely audio (as compared to vision, for example, in which multiple signals can more easily co-exist). And in fact, people talk at the same time quite often, e.g., for barge-in (interrupting someone while they are talking) or verbal backchanneling (e.g, saying "Uh-huh" while someone else is talking).

We have designed, implemented and tested a new multi-threaded real-time architecture, called DiscoRT (Disco for Real-Time) to support the continuous mutual signaling model for embodied conversational agents. Disco is the name of the collaborative discourse (dialogue) manager that is a key component of the architecture—see Section V.

After first discussing related work in the next section, we will briefly introduce the Always-On project,[2] which motivated the development of DiscoRT. In this project, we are using DiscoRT to develop a conversational agent, with both virtual and robotic embodiments, to provide social support for isolated older adults. Next we will focus on nine key use cases the guided our design. Following that, we will describe the implemented architecture and explain how it supports each of

*Current address: Toyota Technological Institute at Chicago, Chicago, IL, USA 60637. Email: bahador.nooraei@gmail.com

[1] Herbert Clark's keynote address at the AAAI 2010 Fall Symposium on Dialogue with Robots, titled "Talk and Its Timing," explained this well and strongly motivated this work.

[2] See http://www.cs.wpi.edu/~rich/always

the use cases. Finally, in the conclusion, we will share our experience thus far in using the architecture.

II. Related Work

Broadly speaking, this work falls into the technical area of "behavior coordination mechanisms"—see review articles by Pirjanian [2] and by Scheutz and Andronache [3]. In particular, we are using what Pirjanian calls a priority-based arbitration model. The seminal example of such a model is Brooks' subsumption architecture [4]. Such models were a response to the observation that agents often have multiple and sometimes conflicting goals, i.e., goals that require different and incompatible behaviors.

We also use Arkin's [5] concept of parallel perception and action *schemas*, which is motivated in part by brain theory and psychology.

The Petri-net synchronization mechanism described in Section V-E is modeled on the Behavior Markup Language (BML) [6] developed for scripting the animation of the first generation of embodied virtual agents.

Other researchers whose work is most similar to ours include Thorisson [7], Bohus and Horvitz [8], and Chao and Thomaz [9]. Thorisson's architecture is more subsumption-like than ours, with three layers of "feedback loops" (schemas) in a fixed priority, whereas our system has only two layers and arbitrates between schemas with variable priorities. Bohus and Horvitz incorporate Bayesian statistics into their model, which we do not. Chao and Thomaz use Petri nets, as we do, to implement synchronization between multimodal behaviors, but do not support arbitration between conflicting behaviors. None of these systems includes a dialogue manager similar to Disco.

Finally, an important real-time issue that our architecture does not directly address is incrementality, especially with respect to natural language processing. For example, as soon as someone says, "the red ball...", before they finish the rest of the sentence, hearers will start to direct their gaze toward a red ball if it is easily visible in the immediate environment. Scheutz [10] and Traum et al. [11] have implemented this kind of incremental processing for interaction with robots and virtual agents.

III. The Always-On Project

Before presenting the use cases for DiscoRT below, we first briefly describe the Always-On project to provide a source of some concrete behavioral examples.

The Always-On project is a four-year effort, currently in its fourth year, supported by the U.S. National Science Foundation at Worcester Polytechnic Institute and Northeastern University. The goal of the project is to create a relational agent that will provide social support to reduce the isolation of healthy, but isolated older adults. The agent is "always on," which is to say that it is continuously available and aware (using a camera and infrared motion sensor) when the user is in its presence and can initiate interaction with the user, rather than, for example requiring the user to log in to begin interaction. Our goal is for the agent to be a natural, human-like presence that "resides" in the user's dwelling for an extended period of time. Beginning

Figure 2. Embodied conversational agent in Always-On project.

in the winter of 2014, we will be placing our agents with about a dozen users for a month-long, four-arm, evaluation study.

Figure 3. Reeti.

We are experimenting with two forms of agent embodiment. Our main study will employ the virtual agent Karen, shown in Figure 2, that comes from the work of Bickmore et al. [12] Karen is a human-like agent animated from a cartoon-shaded 3D model. She is shown in Figure 2 playing a social game of cards with the user. Notice that user input is via a touch-screen menu. Also, the speech bubble does not appear in the actual interface, which uses text-to-speech generation. We are also planning an exploratory study substituting the Reeti[3] tabletop robot, shown in Figure 3, for Karen but otherwise keeping the rest of the system as much the same as possible.

In total, the conversational agent or robot can interact with the user in more than ten different activities including: discuss the weather, learn about the agent, play a social game of cards, talk about family/friends, record a life story to the agent, promote exercise, promote nutrition, hear a humorous tale from the agent, get health tips from the agent, speak with a friend/family member via Skype™ (with all the details of Skype managed by the agent), and manage a personal calendar for the user.

A typical interaction with the agent might start with some greetings (specific to the time of day) and then some discussion of the weather. The weather discussion can be as short as today's weather forecast or extend to the next day, weather in other cities, and weather where friends or family live. At the user's choice, weather might be followed by a social game of cards where the agent's and user's hands in the game and the way the game is played out are commented upon. If the user and agent are somewhat well acquainted, thereafter might follow discussion of the user's family and friends.

[3] See http://www.reeti.fr

IV. The Use Cases

The evaluation of software tools or middleware can be problematic for experimentally-focused disciplines, such as human-computer interaction. Straightforward application of the controlled experiment methodology suggests having two parallel development efforts implementing the same system requirements (one using the new software and one using something else) and then comparing measures of both the development process, such as time spent, and the performance of the resulting systems. Unfortunately, this approach is seldom practical, especially if the tool is designed for professional developers working on large, challenging systems. A experiment with a small, simple system is not likely to show the benefits of the tool.

We have therefore adopted a methodology from software engineering called *use cases* [13], which is particularly well-suited to the domain of interactive systems. In this approach, one identifies at the beginning of the design process a collection of archetypal behaviors (the "use cases") that the system should support and then evaluates the implemented system in terms of how well it supports each of these behaviors. Furthermore, the use cases should be chosen to cover the most challenging aspects of the system's required behavior.

We present nine such use cases for DiscoRT below and then return in Section VI to evaluate how our middleware architecture supports each of them. Notice that the design of DiscoRT aims to support virtual agents and robots that, unlike in the Always-On project, can use their hands and arms to point at and manipulate objects in their environment. DiscoRT is also designed to support fully spoken interaction, as well as the menu-based interaction mode of the Always-On project.

A. Engagement Use Cases

The first group of six use cases for DiscoRT concern supporting *engagement* between the agent and the user. As defined by Sidner et al. [14], engagement is "the process by which two (or more) participants establish, maintain and end their perceived connection during interactions they jointly undertake." Furthermore, as we will see below, engagement involves both verbal and (a lot of) nonverbal behavior.

Case 1) Walking Up to the Agent: The first use case relates to establishing engagement. Specifically, when the user walks by and triggers the motion detector, the agent should awake out of its quiescent state and issue a greeting, such as "good morning." If the user then responds by approaching the computer (which the agent can notice with face detection software on the webcam), the agent should continue start face tracking (see next use case) and continue with a further engaging utterance, such as "how did you sleep?"

The next four use cases relate to maintaining engagement. In later work, we identified four types of what they called "connection events" that function to maintain engagement [15], [16]. We have a use case corresponding to each type of connection event. In general, these use cases involve crucial timing constraints between the verbal and/or nonverbal behaviors of the user and the agent.

Case 2) Face Tracking: Face tracking is the agent's attempt to achieve what is technically called *mutual facial gaze* [17] with the user. When the agent is face tracking, it should orient its gaze toward where it detects the user's face. In addition to being the agent's default gaze behavior for maintaining engagement, mutual facial gaze can have other interaction functions. For example, it is typical to establish mutual facial gaze at the end of a speaking turn (see next use case).

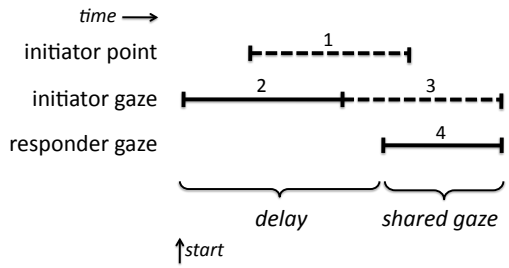

Figure 4. Time line for directed gaze.

Case 3) Turn-Taking: Even though, as discussed in Section I, a conversational interaction entails much more than turn-taking, an embodied conversational agent nevertheless does need to manage speaking turns, particularly in a menu-based system. In linguistics, an *adjacency pair* [18] is the term used to refer to two utterances by two speakers, with minimal overlap or gap between them, such that the first utterance provokes the second utterance. A question-answer pair is a classic example of an adjacency pair. Thus, after producing the so-called "first turn" of an adjacency pair, the agent should wait until the user responds (or until some specified timeout occurs). In some conversational circumstances, the user's response can also be followed by a "third turn" in which the agent, for example, acknowledges the user's response.

Importantly, we generalize the concept of adjacency pair beyond the traditional linguistic definition to include both verbal and nonverbal responses. So for example, a nod can be the answer to a question, instead of a spoken "yes," or the performance of an action can be the nonverbal response to a verbal request, such as, "please pass the salt." Adjacency pairs, of course, also often overlap with the other nonverbal behaviors, such as face tracking and directed gaze (see Use Case 5).

Case 4) Backchanneling: A *backchannel* is an interaction event in which a listener directs a brief verbal or nonverbal communication back to the speaker *during* the speaker's utterance. Typical examples of backchannels are nods and/or saying "uh, huh." Backchannels are typically used to communicate the listener's comprehension of the speaker's communication (or lack thereof, e.g., a quizzical facial expression) and/or desire for the speaker to continue. A conversational agent should be able to both generate appropriate backchannels and interpret backchannels from the user. For example, in the Always-On life-story recording activity, the agent should nod appropriately, even though it does not understand the content of the story that being recorded.

Case 5) Directed Gaze: Finally, Figure 4 shows the time line for the last, and most complex, engagement maintenance use case, called *directed gaze* [19]. In this behavior, one person (the *initiator*) looks and optionally points at some object in the immediate environment in order to make it more salient,

following which the other person (the *responder*) looks at the same object. This behavior is often synchronized with the initiator referring to the object(s) verbally, as in "now spread the *cream cheese* on the *cracker*" (pointing first to the cream cheese and then to the cracker). By turning his gaze where directed, the responder intends to be cooperative and thereby signals his desire to continue the interaction (maintain engagement).

In more detail (see Figure 4), notice first that the act of pointing (1), if it is present, begins after the initiator starts to look (2) at the object. (This is likely because it is hard to accurately point at something without looking to see where it is located.) After some delay, the responder looks at the salient object (4). The initiator usually maintains the pointing (1), if it is present, at least until the responder starts looking at the object. However, the initiator may stop looking at the object (2) before the responder starts looking (4), especially when there is pointing. (This is often because the initiator looks at the responder's face, assumedly to check whether the responder has directed his gaze yet.) Finally, there may be a period of shared gaze, i.e., a period when both the initiator (3) and responder (4) are looking at the same object.

Case 6) Walking Away: The last use case in this section relates to ending engagement. Hopefully, most of the time disengagement between the agent and user will occur as the result of an explicit leave-taking conversational exchange, such as, "Goodbye; See you later." However, the agent should also be prepared to deal with the user simply walking away at any time.

B. Interruption Use Cases

The remaining three use cases relate to various kinds of interruption behaviors. The ability to do more than one thing at a time and smoothly shift between them is a key requirement for a natural conversational agent.

Case 7) Scheduled Event: One reason for interrupting an activity is to due the (imminent) occurrence of a scheduled event. For example, in the Always-On project, the agent helps the user keep a personal calendar of events such as lunch dates, doctor appointments, etc. If the user has a lunch date at noon, the agent should interrupt whatever the agent and user are doing together (e.g., playing cards) ten or fifteen minutes before noon to remind the user of the lunch date and to wrap up or postpone the current activity.

Case 8) Changing Topic: A conversational agent should be able to, either of its own volition, or in response to the user's behavior, smoothly change the topic of the conversation and then, if appropriate, smoothly return to the original interrupted topic. For example, in the Always-On project, activities such as playing cards are viewed as social "containers," within which other topics can also be discussed. In one of our target scenarios, at the end of the user's turn in a card game, the agent says, "By the way, have you thought about my suggestions for how to get more exercise?" After a short discussion of exercise, the agent returns to the card game by saying, "Ok, so I think it's your turn now."

Case 9) Barge-In: Barge-in is a common conversational phenomenon similar to backchanneling (Case 4), in that the listener starts communicating before the speaker's turn is finished. In the case of barge-in, however, the listener's intention is for the speaker to stop and let the listener "take" the turn. A conversational agent should be able to respond to the user's barge-in by promptly ceasing to talk. In a purely spoken language system, the user can barge in simply by starting to speak. A menu-driven system, such as the Always-On project, can support user barge-in by displaying a menu for the user to click on while the agent is still speaking.

The case of a conversational agent barging in on the user is less common. However, Chao and Thomaz [20] describe a strategy, called *minimum necessary information*, for a robot to start acting before the user has finished speaking instructions.

V. System Architecture

Figure 5 shows the architecture of the DiscoRT system that we have designed, implemented and used in the Always-On project to support the use cases above in a principled and general way. Four key features of this architecture are:

1) Multiple Threads: Supporting the continuous mutual signaling model of interaction obviously requires a highly parallel architecture with multiple threads. Each input and output modality and the internal decision making processes needs to function independently without blocking one other.

2) Resource Arbitration: We think of the agent's face, voice, hands and gaze as resources that can be used for different (and sometimes competing) purposes in an interaction. For example, the agent's gaze can be used to achieve mutual facial gaze (Case 2) or it can be used direct the user's gaze to a salient object in the environment (Case 5). Similar to a computer operating system, one of the key functions of DiscoRT is to arbitrate between competing demands for a given resource.

3) Real-Time Control: Timing is critical in all of the use cases. However, there is more margin for error in some use cases than in others. For example, an inappropriate delay of even a fraction of a second in the agent's response to a user-initiated directed gaze or barge-in will be noticeable and degrade the believability of the agent. On the other hand, changing topics or reminding the user of an upcoming scheduled event can be delayed a second or two without harmful effect. We call these "hard" and "soft" real-time constraints, respectively, and handle them separately in the architecture.

4) Dialogue Management: A unique feature of our architecture is its integration with the Disco dialogue manager. Among other things, Disco provides a focus stack for helping to manage interruptions.

We return now to Figure 5 for a high-level tour of the architecture, following which we will discuss several aspects in more detail.

Starting at the left of the figure, we see the *perceptors*, which implement the input side of the system's multimodal interface with the user. At the bottom of the figure are the resources which the system needs to control/manage, such as the agent's face, voice, hands, gaze and so on (the specific resources may vary between agents). Notice that Disco's focus stack is also viewed as a resource; the implications of this will be discussed below. Along the top of the figure are an

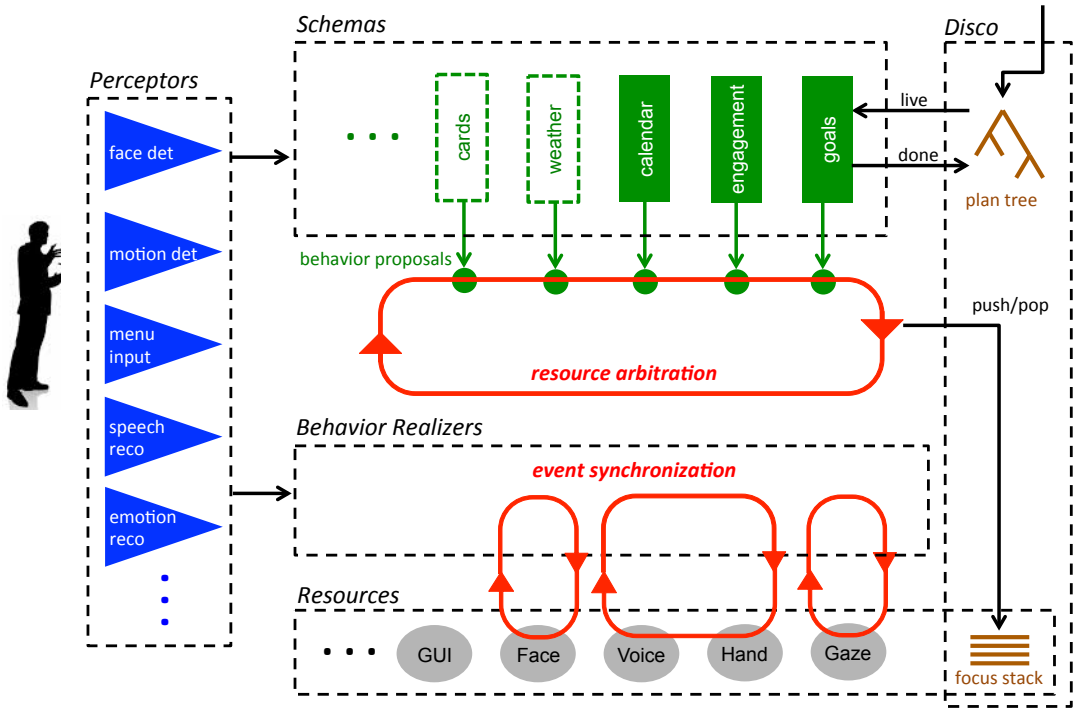

Figure 5. DiscoRT system architecture.

open-ended collection of *schemas*, which in parallel make proposals for behaviors using particular resources. Resource conflicts between behavior proposals are arbitrated by the *resource arbitration* (soft) real-time process—the loop in the middle of the figure. Each behavior that survives the arbitration process results in instantiation of a *realizer* process(the loops in the figure that intersect the resources) that handles the (hard) real-time synchronization of behavioral events involving the assigned resource(s). Finally, the right side of the figure shows the Disco dialogue manager, which has a plan tree and focus stack as its two main data structures.

A. Perceptors

Perceptors are the system's abstraction for sensing capabilities, such as face detection, motion detection, menu input, speech recognition, and so on. The specific perceptors may vary between agents, depend on the hardware and software configuration of the agent.

Some perceptors simply abstract the output of a single hardware sensor, such as the infrared motion detector in the Always-On project. However, perceptors can also fuse information from multiple input modes, such as an emotion recognition perceptor that combines facial expression information from a camera with tone of voice information from a microphone. A single item of hardware, such as a camera, may also serve multiple perceptors, such as face detection and gaze tracking. In the Always-On project, the touch-screen menu is also modeled as a perceptor.

Each perceptor runs on its own thread with a loop cycle rate depending on its needs, such as the frame rate of the camera. The output of the perceptors is used by both the schemas and the realizers (discussed in more detail below). For example, face detection and motion perception are used by the schema that establishes engagement. Gaze tracking is used by the realizer that implements the direct-gaze behavior discussed in Case 5. Perceptors support both polling and event-based API's.

B. Schemas

Schemas are the core of the DiscoRT architecture. In the Always-On project there is a schema corresponding to each social activity that the agent can do with the user, such as talking about the weather, playing cards, etc. These schemas are created and destroyed as the system runs. A few other schemas, such as the schema that manages engagement, are always running.

The fundamental function of a schema is to continually propose *behaviors*. Each schema runs on it own thread with a loop cycle rate depending on its needs and typically maintains its own internal state variables. In order to decide what behavior to propose at a given moment (it is allowed to propose no behavior), a schema can consult its own state variables and the system's perceptors, as well as external sources of information. For example, the weather schema downloads up-to-date weather information from the internet.

Conceptually, a behavior specifies a "program" (called the *realizer*—see Section V-E) to be executed using one or more resources. For example, when a schema wants to "say" something, it proposes a text-to-speech behavior that specifies the utterance string and requires the voice resource. A more complex example is the directed gaze behavior (see Figure 4), which requires three resources (voice, hand and gaze) and includes a complex realizer program to synchronize the control of the resources with input from the perceptors.

Some schemas, such as for talking about the weather, are purely conversational, (see Section V-D), and some, such as playing cards, also involve the manipulation of shared artifacts, such as cards. In the case of a virtual agent, such manipulations use the *GUI* resource. In the Always-On project, playing cards is a social activity, that involves both manipulation and conversation about the game, e.g., "I've got a terrible hand."

There are no restrictions in DiscoRT on the internal implementation of a schema. In the Always-On project we have primarily implemented schemas as state machines and using D4g [21], which is an enriched formalism for authoring dialogue trees.

C. Resource Arbitration

The resource arbitration thread/loop runs approximately once per second and gathers up the collection of behavior proposals from all of the running schemas. Proposals with non-overlapping resource requirements are directly scheduled, i.e., an instance of the specified realizer is created and started running (unless it is already running). Resource conflicts between proposals are resolved using a simple system of per-schema priorities with some fuzzy logic rules [22] to make the system more stable, i.e., to prevent switching behaviors too quickly between closely competing schemas. The behavior proposals that are chosen are then scheduled.

D. Dialogue Management

Since DiscoRT is designed to support conversational agents, it includes specialized machinery for dialogue management. We are using the Disco dialogue manager, which is the open-source successor to Collagen [23], [24]. Disco has two key two key data structures: a plan tree and a focus stack. Each of these has a point of integration with DiscoRT, as shown in Figure 5.

The *plan tree*, which is typically provided from another component outside of DiscoRT, represents the agent's goals for its interaction with user as a hierarchical task network [25]. This formalism includes optional and repeated goals and partial ordering constraints between subgoals. Thus the typical interaction example, starting with greetings, etc., described in Section III above is formalized as a plan tree. The plan tree can be updated while the system is running.

DiscoRT includes a predefined schema, called the *goals* schema in Figure 5, that is always running and automatically starts the schema(s) corresponding to the currently live goal(s) in the plan tree. Thus, for example, when the "discuss the weather" goal becomes live, the goals schema starts the weather schema. When the schema exits, the corresponding goal in the plan tree is automatically marked as done.

The *focus stack* is stack of goals[4], which captures the familiar phenomenon of pushing and popping topics in human conversation. For example, in the middle of a card game, the agent might temporarily suspend (push) playing cards to talk about exercise or to remind the user about an upcoming appointment in her calendar, and then return (pop) to playing cards.

[4]Technically, a stack of focus spaces, each of which includes a goal, called the discourse segment purpose; but this is beyond the scope of this paper.

In DiscoRT, this kind of interaction is achieved by making the focus stack a resource that represents control of the current topic of conversation. Schemas that involve conversation, such as the cards schema, the exercise schema, and the calendar schema in the example above, require the focus stack in their behavior proposals. When the resource arbitrator starts a new behavior realizer that requires the focus stack, it pushes the goal associated with the proposing schema onto the stack (unless it is already there). When the behavior realizer finishes, the goal is automatically popped off the stack. (See Case 7 in Section VI for detailed example.)

This integration between the dialogue model and the schema architecture in DiscoRT is very powerful and flexible. For example, it is possible and sometimes useful for a schema to propose a speech behavior *without* requiring the focus stack. If you poke a robot, it might respond by saying "Ouch!" without changing the conversational focus. We are also experimenting with providing hooks in DiscoRT for automatically producing generic transition language when the stack is pushed and popped, such as "excuse the interruption, but…" or "now, returning to…"

E. Behavior Realizers

The behavior realizers (bottom of Figure 5) implement the hard real-time event synchronization in the system, such as responding to user-initated directed gaze. A behavior realizer in DiscoRT is very similar (hence the name) to a BML realizer. However, for the reasons discussed in detail in [26], we use an event-driven Petri net rather than a fixed schedule as in most BML realizers.

Further, in DiscoRT there can be be multiple realizers independently controlling non-overlapping resource sets (think of a robot "rubbing its tummy and patting its head" at the same time). Each behavior realizer has a separate thread that runs by default at 10Hz. Realizers often get information from perceptors.

A simple realizer, such as for a smile behavior, uses only one resource (e.g., face) and does not get information from perceptors (e.g., it just waits until the smile is completed). More complex realizers, such as for a directed gaze behavior, use multiple resources (e.g., voice, hand and gaze) and access one or more perceptors (e.g., hand and gaze tracking) to implement multimodal synchronization.

A realizer starts execution when the resource arbitrator accepts a schema's behavior proposal as described above. If the schema stops proposing that behavior, then the realizer is automatically stopped. The realizer program can also stop itself (e.g., if has completed the behavior), in which case the proposing schema is notified, so that it can update its internal state for making future proposals.

Our realizers support all of the essential timing relationships of BML (synchronize, before, after and fixed delay). We did not implement the complete BML specification because much of it concerned specific gestural actions; we wanted a more general framework. Also, our realizer programming API uses Java rather than XML.

VI. Evaluation of the Use Cases

In this section, we describe how each of the use cases defined in Section IV is handled by the DiscoRT architecture.

Case 1) Walking Up to the Agent: The engagement schema (implemented as a state machine) continually polls the motion perceptor (which abstract the IR motion detector hardware). When motion is detected, the schema proposes a speech behavior (e.g., "Good morning") and enters a state in which it starts polling the face perceptor (which abstracts the output of face detection software operating on the webcam output). When a face is detected, the schema proposes another speech behavior and enters the *engaged* state; otherwise, after a timeout, the schema returns to the motion perceptor polling state.

Case 2) Face Tracking: Face tracking is implemented by a behavior realizer that requires the gaze resource and uses the face detection perceptor. The realizer simply runs a loop that updates the agent's gaze to where it currently sees the user's face. The face tracking behavior that causes the realizer to be started is proposed by the engagement schema when it enters the engaged state (see Case 6 for stopping the realizer).

Case 3) Turn-Taking: Turn-taking is implemented by an abstract state machine that is reused in all of the schemas that include conversational behavior, such as weather, cards and calendar. The agent's utterances are produced by proposing speech behaviors. In menu-based systems, the state machine waits for the user's response by waiting for an event from the menu perceptor (which abstracts the menu GUI) or the speech perceptor (which abstracts speech recognition).

Case 4) Backchanneling: Backchanneling is a hard real-time phenomenon, so it must be implemented by a behavior realizer that receives events from a perceptor that detects appropriate moments at which to produce backchannels, such as the end of phrases or sentences in the user's speech. The schema that proposes the backchanneling behavior is responsible for deciding what form of backchannel should be used (e.g., positive or negative) and for re-proposing a new behavior when the form of backchannel should be changed.

Case 5) Directed Gaze: As discussed in Section V-E, the time line for directed gaze (Figure 4) is implemented as a behavior realizer. Any schema can propose a directed gaze behavior.

Case 6) Walking Away: The engagement schema state machine includes a timeout to notice when there has been no face or motion perceived. When this occurs, it stops proposing face tracking and enters the waiting for engagement state described in Case 1.

Case 7) Scheduled Event: While the user and agent are playing cards together, both the card schema and the calendar schema are running, but the calendar schema is not making any behavior proposals. The card schema's behavior proposals require the focus stack, so the card playing goal stays on the top of the stack. Then, triggered by the clock time, the calendar schema proposes a speech behavior that requires the focus stack. Because the calendar schema has a higher priority, the arbitrator gives it control of the focus stack, which causes the calendar reminder goal to be pushed on top of the card playing goal. The calendar reminder goal remains on the top of the stack throughout the (sub)dialogue regarding the upcoming appointment. When this reminder dialogue is completed, the calendar schema stops making proposals, the calendar goal is popped, and the arbitrator gives the focus stack resource back to the card schema, which has been continuously proposing the next behavior in the game, but never getting the needed focus stack resource.

Case 8) Changing Topic: Changing topic is handled similarly to Case 7, except that instead of the calendar schema deciding to make the interruption, the decision that the new topic has a higher priority than the current topic is made by some other schema based on cognitive reasoning that is outside of the scope of DiscoRT. The mechanism of pushing and popping the focus stack is identical, however. Furthermore, the interrupted schema may stop proposing the old topic, in which case it is never returned to.

Case 9) Barge-In: Barge-in is handled in the speech behavior realizer, which in addition to controlling the text-to-speech engine (voice resource), listens for events from the menu or speech perceptor (depending on the type of system). When such an event is received, the realizer immediately stops the text-to-speech engine and terminates.

VII. Conclusion

DiscoRT has succeeded in its design goal of supporting the specified use cases in a principled and general way and has been extremely useful in implementing the Always-On project. Furthermore, we feel that the use cases themselves are a research contribution towards evaluating other systems for similar purposes. Our only negative report on DiscoRT is that it has been challenging to master the schema-based programming approach. This is not entirely surprising, since programming highly-parallel systems is well-known to be difficult. DiscoRT is implemented in Java and is available from the authors under an open-source license.

Acknowledgment

This work is supported in part by the National Science Foundation under award IIS-1012083. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] C. Rich and C. Sidner, "Robots and avatars as hosts, advisors, companions and jesters," AI Magazine, vol. 30, no. 1, 2009.

[2] P. Pirjanian, "Behavior coordination mechanisms: State-of-the-art," Inst. for Robotics and Intelligent Systems, U. of Southern California, Tech. Rep. IRIS-99-375, 1999.

[3] M. Scheutz and V. Andronache, "Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems," IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, no. 6, 2004, pp. 2377–2395.

[4] R. Brooks, "A robust layered control system for a mobile robot," IEEE J. of Robotics and Automation, vol. 2, 1986, pp. 14–23.

[5] R. Arkin and D. MacKenzie., "Temporal coordination of perceptual algorithms for mobile robot navigation," Robotics and Automation, vol. 10, no. 3, 1994, pp. 276–286.

[6] S. Kopp et al., "Towards a common framework for multimodal generation: The Behavior Markup Language," in Proc. Int. Conf. on Intelligent Virtual AgentsIntelligent Virtual Agents, 2006.

[7] K. Thorisson, "Natural turn-taking needs no manual: Computational theory and model, from perception to action," Multimodality in Language and Speech Systems, vol. 19, 2002.

[8] D. Bohus and E. Horvitz, "Models for multiparty engagement in open-world dialog," in Proceedings of the SIGDIAL 2009 Conference. London, UK: Association for Computational Linguistics, September 2009, pp. 225–234.

[9] C. Chao and A. Thomaz, "Timing in multimodal turn-taking interactions: Control and analysis using timed petri nets," J. Human-Robot Interaction, vol. 1, no. 1, 2012, pp. 4–25.

[10] E.Krause, R. Cantrell, E. Potapova, M. Zillich, and M. Scheutz, "Incrementally biasing visual search using natural language input," in Proc. 12th Int. J. Conf. on Autonomous Agents and Multiagent Systems, St. Paul, MN, 2013.

[11] D. Traum, D. DeVault, J. Lee, Z. Wang, and S. Marsella, "Incremental dialogue understanding and feedback for multiparty, multimodal conversation," in Proc. Int. Conf. on Intelligent Virtual Agents, Santa Cruz, CA, 2012, pp. 245–251.

[12] T. Bickmore, L. Caruso, and K. Clough-Gorr, "Acceptance and usability of a relational agent interface by urban older adults," in Proc. ACM Conf. on Computer Human Interaction. ACM, 2005, pp. 1212–1215.

[13] D. Leffingwell and D. Widrig, Managing Software Requirements: A Use Case Approach (Second Edition). Addison-Wesley, 2012.

[14] C. L. Sidner, C. Lee, C. Kidd, N. Lesh, and C. Rich, "Explorations in engagement for humans and robots," Artificial Intelligence, vol. 166, no. 1-2, 2005, pp. 104–164.

[15] C. Rich, B. Ponsler, A. Holroyd, and C. Sidner, "Recognizing engagement in human-robot interaction," in Proc. ACM Conf. on Human-Robot Interaction, Osaka, Japan, Mar. 2010.

[16] A. Holroyd, C. Rich, C. Sidner, and B. Ponsler, "Generating connection events for human-robot collaboration," in IEEE Int. Symp. on Robot and Human Interactive Communication, Atlanta, GA, 2011.

[17] M. Argyle and M. Cook, Gaze and Mutual Gaze. New York: Cambridge University Press, 1976.

[18] D. Crystal, The Cambridge Encyclopedia of Language. Cambridge, England: Cambridge University, 1997.

[19] A. Kendon, "Some functions of gaze direction in two person interaction," Acta Psychologica, vol. 26, 1967, pp. 22–63.

[20] C. Chao, C. Lee, M. Begum, and A. Thomaz, "Simon plays simon says: The timing of turn-taking in an imitation game," in Int. Symp. on Robot and Human Interactive Communication (RO-MAN), 2011, pp. 235–240.

[21] C. Rich and C. L. Sidner, "Using collaborative discourse theory to partially automate dialogue tree authoring," in Proc. Int. Conf. on Intelligent Virtual Agents, Santa Cruz, CA, Sep. 2012.

[22] J. Fodor and M. Roubens, Fuzzy Preference Modeling and Multicriteria Decision Support. Kluwer Academic, 1994.

[23] C. Rich and C. Sidner, "Collagen: A collaboration manager for software interface agents," User Modeling and User-Adapted Interaction, vol. 8, no. 3/4, 1998, pp. 315–350, reprinted in S. Haller, S. McRoy and A. Kobsa, editors, *Computational Models of Mixed-Initiative Interaction*, Kluwer Academic, Norwell, MA, 1999, pp. 149–184.

[24] C. Rich, C. Sidner, and N. Lesh, "Collagen: Applying collaborative discourse theory to human-computer interaction," AI Magazine, vol. 22, no. 4, 2001, pp. 15–25, Special Issue on Intelligent User Interfaces.

[25] C. Rich, "Building task-based user interfaces with ANSI/CEA-2018," IEEE Computer, vol. 42, no. 8, Aug. 2009, pp. 20–27.

[26] A. Holroyd and C. Rich, "Using the behavior markup language for human-robot interaction," in Proc. ACM Conf. on Human-Robot Interaction, Boston, MA, Mar. 2012.