DTask & LiteBody: Open Source, Standards-based Tools for Building Web-deployed Embodied Conversational Agents

Timothy Bickmore, Daniel Schulman, and George Shaw

Northeastern University College of Computer & Information Science,
360 Huntington Ave, WVH202, Boston, MA 02115, USA
{bickmore,schulman,shaw}@ccs.neu.edu

Abstract. Two tools for developing embodied conversational agents and deploying them over the world-wide web to standard web browsers are presented. DTask is a hierarchical task decomposition-based dialogue planner, based on the CEA-2018 task description language standard. LiteBody is an extensible, web-based BML renderer that runs in most contemporary web browsers with no additional software and provides a conversational virtual agent with a range of conversational nonverbal behavior adequate for many user-agent interaction applications. Together, these tools provide a complete platform for deploying web-based conversational agents, and are actively being used on two health counseling applications.

Keywords: Dialogue planning, embodied conversational agent, relational agent, open source, behavior markup language.

1 Introduction

There is a growing interest in building common, standards-based software frameworks to support Embodied Conversational Agent (ECA) development, such as the SAIBA/BML/FML standardization efforts [8]. The motivation for this work is to reduce duplication of effort in developing new systems, and to enable modules from different developers to be assembled into working systems. The approach taken in much of this work is to identify a core set of common functionality, but then to define a framework in which all possible ECA functions and future extensions can be accommodated.

In contrast, for many applications, developers only need a tiny subset of this functionality. An example is an application in which there is only one ECA and it only talks directly to the user while displaying a well-defined range of nonverbal conversational behavior. Examples of such systems include pedagogical agents [6], health counseling agents [2, 3], and direction-giving agents [4]. For such applications, many existing tools represent significantly more overhead and developer learning time than should be necessary. Further, such applications are constrained enough in their functionality that they have the potential to be deployed over the web using standard browsers without additional software, greatly increasing the possibility of wide dissemination of ECAs built this way.

Our approach has been to develop a minimal ECA framework for this class of applications, while still adhering to available standards as much as possible. The current result of our effort is two tools—a dialogue engine and an ECA renderer—which, together, provide a complete framework in which user interface ECAs with rich dialogue content can be built and immediately deployed over the web. In addition, because they are based on several standards, these tools should facilitate sharing of dialogue, character animation, and user interface content among researchers and application developers.

2 The DTask Dialogue Engine

DTask is a dialogue planner designed to model and execute system-directed dialogue, with multiple-choice user input. Dialogue is specified declaratively, as a hierarchical task decomposition. The DTask application functions as a network server and relies on a user interface client to provide an interface, such as an ECA, to the user. In our current work, LiteBody (described in Section 3) provides this client functionality, and includes a BML-driven ECA.

2.1 CEA-2018

Following Shared Plans theory [7, 9], we treat dialogue as a collaboration in which participants coordinate their action towards achieving a shared goal. The intentional structure of dialogue is modeled as a hierarchical task decomposition: sets of recipes (goal decompositions) and subtasks which may be used to achieve the overall goal of the collaboration. Dialogue context is modeled as a runtime focus stack, representing the subgoals currently adopted. DTask's hierarchical task model is an implementation of the ANSI/CEA-2018 standard [10], which is itself inspired by the COLLAGEN dialogue engine [11]. CEA-2018 specifies an XML-based representation for a set of tasks, and a set of recipes which can be used to achieve those tasks. A recipe describes one way of decomposing a goal into a partially-ordered set of subgoals and/or primitive actions. ECMAScript [1] is used to specify task preconditions and postconditions, recipe applicability conditions, constraints on task parameters, and grounding of atomic tasks.

DTask extends CEA-2018 with a mechanism for declarative specification of dialogue. A turn of dialogue is specified as an adjacency pair template (APT): an agent utterance and a list of possible user responses, which comprise a primitive action in the task description. An APT achieves a particular task in the task model; there may be an arbitrary number of APTs, as well as recipes, which can achieve a task.

The surface form of agent and user utterances may include both natural language, and variables to be filled in at the time the utterance is produced. Generally, nonverbal behavior of the ECA is not specified in the dialogue model, with the intent that it be added automatically at runtime by a system such as BEAT [5]. However, explicit annotations can be added in BML, or in any other format understood by the user interface client.

2.2 DTask Example

Fig. 1 shows an example of a fragment of a DTask dialogue model. This example implements a task ("RitualIntro") which is a short ritualized greeting to the user. There is one recipe provided which can achieve this task ("DoRitualIntro"), consisting of two subtasks ("HowAreYou" and "RespondToIntro").

Tasks can have locally-scoped input and output parameters. The "HowAreYou" task has one output slot, which defines the semantics of the user's utterance which will be represented. In this case, the semantics is simply a boolean value representing whether the user asked a reciprocal question of the agent.

The "RespondToIntro" task has one input parameter, and constraints on the recipe are used to bind the output of the earlier task to this input. The example shows one dialogue turn which can be used to achieve this task in the case where the user requested a reciprocal response. There may be any number of other dialogue turns specified with different applicability conditions.

```
<task id="RitualIntro"/>
<subtasks id="DoRitualIntro" goal="RitualIntro">
   <step name="ask" task="HowAreYou"/>
   <step name="respond" task="RespondToIntro"/>
   <binding slot="$respond.reciprocal" value="$ask.reciprocal"/>
</subtasks>
<task id="HowAreYou">
   <output name="reciprocal" type="boolean"/>
   <d:turn>
      <d:agent>Hi {USER.name}. How are you?</d:agent>
      <d:user>
         <d:say>I'm good.  How are you?</d:say>
         <d:result slot="reciprocal" value="true"/>
      </d:user>
      <d:user>
         <d:say>Good.</d:say>
         <d:result slot="reciprocal" value="false"/>
      </d:user>
   </d:turn>
</task>
<task id="RespondToIntro">
   <input name="reciprocal" type="boolean"/>
   <d:turn>
      <applicable>$this.reciprocal</applicable>
      <d:agent>Great. Thanks for asking!</d:agent>
…
```

Fig. 1. Example DTask Task Descriptions and Recipes

3 The LiteBody User Interface ECA

Many ECA applications could ideally be fielded on users' home computers where users could interact with the agents at their convenience. However, hardware requirements for 3D graphics and unwillingness to install custom software, especially for occasional use, represent barriers to wide dissemination of many of these ECA applications. LiteBody is a web-enabled, ECA-based user interface which renders an ECA given BML commands from a dialogue engine. This is accomplished, in part, by synthesizing speech on a server and dynamically streaming it to the user's browser as needed. The application also presents the user with a range of input widgets (under control of the dialogue engine) to elicit user contributions to the conversation, and returns input information to the dialogue engine for interpretation.

The ECA provides a range of common conversational nonverbal behavior, including: visemes and eyebrow raises synchronized to speech, head nods, facial displays of emotion, posture shifts, gazing at and away from the user, and idle behavior (blinking, etc.). While the character can appear only in (continuously variable) mid-range to close-up shots facing the user, it can hold up and point at 2D objects (e.g., images, documents, web pages) in front of it. The background behind the character can be any dynamically loadable image (actually any Flash file, including animations). The interface also supports an extensible set of user input widgets, but currently provides multiple choice input buttons (Fig. 2) and a free text input box.

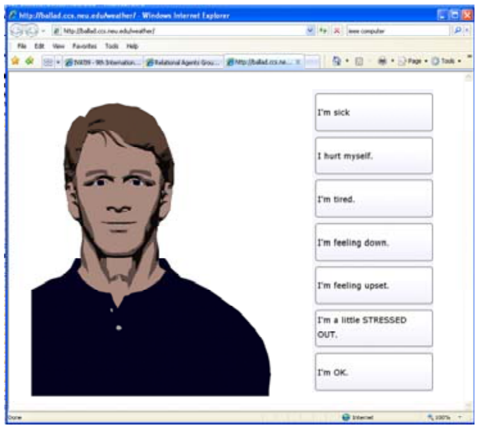

Fig 2. Example LiteBody Character with Multiple Choice User Input Buttons

3.1 LiteBody Architecture

The LiteBody architecture consists of a Dialogue Server, a Text-To-Speech (TTS) Server, and a web-based Flash Client (Fig. 3).

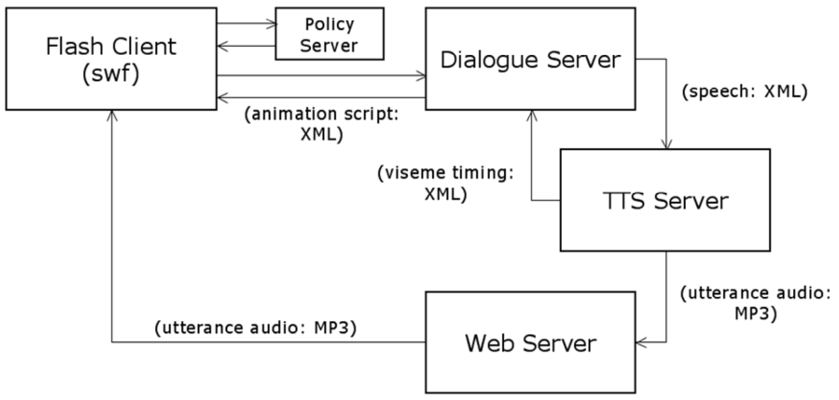

Fig. 3. LiteBody Architecture

The Dialogue Server represents the LiteBody interface point for a dialogue engine or other application controlling the ECA. In response to a single BML command, the Dialogue Server causes the TTS Server to produce a web-accessible mp3-format audio file of the ECA's speech, along with an XML document that contains speech-synchronized timing information for animation and other actions to be performed by the ECA in the Flash Client. The XML document also contains the URL for the audio file that the Flash Client will use to stream the speech mp3 from. Once the audio file has finished production, the XML document is transferred to the Flash Client for execution.

The TTS Server uses any speech synthesizer that is compatible with the Microsoft Speech API (v5.1) to generate mp3 audio files for download to the FlashClient (via any standard web server) along with phoneme and word boundary timing information.

The Flash Client is built entirely in Adobe Flash using the ActionScript programming language and the standard Flash rendering engine for onscreen display. Flash provides a lightweight, near ubiquitous platform on which to deliver dynamic multimedia content via the web.

After being downloaded into the user's browser and initializing, the Flash Client makes a persistent socket connection with the Dialogue Server. Once a socket connection is made, the Dialogue Server may begin sending XML actions to the client for execution. Upon receipt of an action, the client inserts a new ActionObject into its animation queue. An ActionObject, contains any number of "untimed" commands such as loading documents or audio files, as well as any number of "timed" commands that must be synchronized with the speech audio. These timed actions each carry a timestamp in milliseconds relative to the beginning of performance of the action.

When the Flash Client begins a new action by taking the next ActionObject in the queue and sending it to the AnimationEngine, it first performs any untimed commands, such as the loading of a speech audio file. Flash has sophisticated streaming capabilities, which we leverage in order to begin realizing an utterance before the audio file has finished loading to minimize latencies between speaking turns. Once the audio file begins playing, a timer starts and the performance of timed actions begins, synchronized with the playback.

When the AnimationEngine determines that a particular action is due to be performed, it sends a message to the Rendering Engine telling it to make the appropriate changes to the onscreen ECA representation (for animation actions) or to perform other necessary commands (such as loading an external file or document). Animation actions are realized by either moving Flash's playhead to a new frame in the active movie clip, or by swapping the active movie clip for a new one.

Upon completion of all of an ActionObject's actions, a message is sent to the Controller. The Controller relays a message to the Dialogue Server that an action has been completed, and then checks the queue to see if another action is waiting. If not, the Controller either waits for another action to arrive from the server, or generates an idle action such as an eye-blink or a posture shift for the ECA.

User input actions are handled in a similar fashion to other actions, except that there is no timing information associated with a user input, and realization of the input interface is handled via a UserInputEngine class.

Fig. 4 shows an example BML command that LiteBody can execute. In this command, the ECA is being instructed to speak with the audio file URL specified in the "speech" command, and the visemes and their timing relative to the start of speech specified in the "lips" commands. BML extensions ("rag" namespace) are used to load a new background behind the character, modify the camera shot by zooming in to the ECA over the first 300ms of speech, and changing the ECA's facial expression to "happy" between 500ms and 800ms relative to the start of speech.

```
<BML>
<speech id="s1" start="0" type="xxx/mp3"
ref="http://localhost:8080/speech/file0000000001.wav.mp3" text="" />
<rag:background url="data/checkers.swf" />
<rag:zoom time="0" duration="300" timestamp="0" value="0.2" />
<rag:expression time="500" duration="300" timestamp="500" value="happy"/>
<lips viseme="1" time="870" />
<lips viseme="1" time="940" />
<lips viseme="9" time="1140" />
…
</BML>
```

Fig. 4. Example LiteBody BML Command

The Flash Client is designed to be extensible in several ways. Arbitrary user input widgets may be added to elicit information or conversational input from the user in a wide variety of ways. An input widget can be any Flash movie, with the only requirement being that it presents a button to submit its input, and returns the user input as a string to the server.

New animation sequences can also be added to the Flash Client with minimal effort. After adding the animation artwork, a new named keyframe is added that carries the name of the new animation, and the name and duration of the animation is added to a configuration file.

5 Conclusions and Future Plans

Our goals in developing DTask and LiteBody were to make the development of a certain class of conversational virtual humans significantly easier, allow for extensibility and modularity through adherence to public standards, and provide the ability for wide dissemination of developed systems over the web. These tools are currently being used in two health education and health behavior change projects funded by the US National Institutes of Health. We have a commitment to our funding agency to release these tools as open source for the benefit of the virtual human and health informatics research communities, and welcome collaborations on the further development and application of these tools.

Acknowledgments. Thanks to Candace Sidner, Dolphy Fernandes, Rukmal Fernando, Langxuan Yin and the other members of the Relational Agents Group for their help with this work. This project was supported by NIH National Library of Medicine Grant R21LM008995.

References

1. Ecma International, www.ecma-international.org.
2. Bickmore, T., Pfeifer, L., and Paasche-Orlow, M. Health Document Explanation by Virtual Agents *Intelligent Virtual Agents*, Paris, 2007, 183-196.
3. Bickmore, T. and Picard, R. Establishing and Maintaining Long-Term Human-Computer Relationships. *ACM Transactions on Computer Human Interaction*, *12* (2). 293-327.
4. Cassell, J., Stocky, T., Bickmore, T., Gao, Y., Nakano, Y., Ryokai, K., Tversky, D., Vaucelle, C. and Vilhjálmsson, H., MACK: Media lab Autonomous Conversational Kiosk. in *Imagina '02*, (Monte Carlo, 2002).
5. Cassell, J., Vilhjálmsson, H. and Bickmore, T., BEAT: The Behavior Expression Animation Toolkit. in *SIGGRAPH '01*, (Los Angeles, CA, 2001), 477-486.
6. Graesser, A. *et al*, AutoTutor: A simulation of a human tutor. *Cognitive Systems Research*, *1*.
7. Grosz, B. and Sidner, C. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, *12* (3). 175-204.
8. Kopp, S., Krenn, B., Marsella, S., Marshall, A.N., Pelachaud, C., Pirker, H., Thórisson, K. and Vilhjálmsson, H. Towards a Common Framework for Multimodal Generation: The Behavior Markup Language *Intelligent Virtual Agents*, Marina Del Rey, CA, 2006.
9. Lochbaum, K. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics*, *24* (4). 525-572.
10. Rich, C. Building Task-Based User Interfaces With ANSI/CEA-2018. *IEEE Computer (to appear)*.
11. Rich, C. and Sidner, C.L. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, *8* (3-4). 315-350.