A Mark-Up Language and Interpreter for Interactive Scenes for Embodied Conversational Agents

David Novick(✉), Mario Gutierrez, Ivan Gris, and Diego A. Rivera

Department of Computer Science, The University of Texas at El Paso,
500 West University Avenue, El Paso, TX 79968-0518, USA
novick@utep.edu,
{mgutierrezl9,darivera2}@miners.utep.edu,
ivangris4@gmail.com

Abstract. Our research seeks to provide embodied conversational agents (ECAs) with behaviors that enable them to build and maintain rapport with human users. To conduct this research, we need to build agents and systems that can maintain high levels of engagement with humans over multiple interaction sessions. These sessions can potentially extend to longer periods of time to examine long-term effects of the virtual agent's behaviors. Our current ECA interacts with humans in a game called "Survival on Jungle Island." Throughout this game, users interact with our agent across several scenes. Each scene is composed of a collection of speech input, speech output, gesture input, gesture output, scenery, triggers, and decision points. Our prior system was developed with procedural code, which did not lend itself to rapid extension to new game scenes. So to enable effective authoring of the scenes for the "Jungle" game, we adopted a declarative approach. We developed ECA middleware that parses, interprets, and executes XML files that define the scenes. This paper presents the XML coding scheme and its implementation and describes the functional back-end enabled by the scene scripts.

Keywords: Embodied conversational agents · Scene · Interpreter · Parser

1 Introduction

Authoring of scenes in which embodied conversational (ECAs) agents interact with humans currently has limited technological support. Scenes tend to be written directly as a computer program that specifies procedurally how the ECA should behave. As the number and complexity of scenes increases, writing the scenes becomes correspondingly more difficult and more prone to the sorts of problems associated with unstructured code. Scene developers could write better scenes, more quickly, if they had a way to write scenes less like a writing a computer program and more like writing a script for a play.

Writing scenes for ECAs, however, is much more complicated than writing a play because the scenes have to account for the technical details of the agents' inputs and outputs through speech and gesture and the technical details of the agent's virtual world. Ideally, ECAs can interact with human beings in such a way that they require

R. Shumaker and S. Lackey (Eds.): VAMR 2015, LNCS 9179, pp. 206–215, 2015.
DOI: 10.1007/978-3-319-21067-4_22

little to no human intervention to complete tasks and or to navigate conversations. But writing scenes for this is a daunting task, especially when the interaction space involves shared reality, where the virtual side of the interaction contains with which both agent and human can interact). High-functionality ECA systems have to include gesture recognition for task-based processes, gesture recognition for general annotation, branching, 3D movement in virtual space, speech, and other sophisticated features.

The research program of UTEP's Advanced aGEnt eNgagement Team (AGENT) seeks to provide ECAs with behaviors that enable them to build and maintain rapport with human users. To conduct this research, we need to build agents and systems that can maintain high levels of engagement with humans over multiple interaction sessions. These multiple sessions can potentially run over longer periods of time to examine long-term effects of the virtual agent's behaviors, so our research team has to write and implement many scenes.

Our team's most recent system, "Escape from the Castle of the Vampire King" (Gris et al. 2014), had a human and ECA play a game over two sessions. The program, though, was essentially a text-based game, such as Colossal Cave (Crowther et al. 1976) and Zork (Anderson and Galley 1985), realized with an ECA serving as narrator and with game-appropriate scenery. Dialog management could be handled by a relatively simple command interpretation loop. Although development of the "Vampire King" game involved writing a program complex enough to require eventual refactoring, the system was still simple enough that it could be developed with traditional software engineering techniques.

Our current system, "Survival on Jungle Island," is much more complex, though. It has already has eight scenes, and we expect it to have at least eight more, so that the game can provide extended interaction between human and ECA. Consequently, development of "Jungle Island" using the procedural representations and traditional software engineering techniques we used for the "Vampire King" game would have made writing the game's many scenes unreasonably complicated and burdensome. This meant finding a much more efficient way of developing scenes for ECA systems, a way that would enable us to write dozens of scenes with only modest technical effort beyond imagining the scenes' substantive content.

The requirements of developing systems like "Jungle Island" led us to adopt a declarative approach to authoring scenes for human-ECA interaction. In our approach, developers write scenes as scripts, using a mark-up language represented as XML tags, and these scripts are interpreted by middleware to drive the real-time system. In this paper we present our XML-based mark-up language for scripting human-ECA interactions, describe its implementation in middleware, describe the functional back-end enabled by the scene scripts, contrast our approach with existing standards such as SAIBA, FML, and BML, present an excerpt of a representative scene script, discuss the advantages and limitations of our approach, and outline related future work.

2 Scene Mark-Up Language

In this section we review the functionality of our mark-up language and its interpreter. We discuss each tag and its function.

Scripts using the mark-up language rely on twelve XML tags to identify agent behaviors and scene flow. A scene is a session of interaction, typically designed to take four to five minutes, that includes multiple human-ECA exchanges. Each exchange is called an episode, and the flow of the dialog within a scene is managed by choosing different episodes as a function of users' responses.

In developing our ECA system, we aimed for a design that promoted the sensation of immersion in the virtual world and the naturalness of the interaction between human and ECA. This led us to adopt a robust approach that would provide scene developers with control of scene progression by adaptively enabling or disabling speech-recogntion grammars for specific dialogs, gesture recognition libraries for specific gesture event, and control of what happens in the 3D scenes.

Authors specify scenes declaratively with XML tags such as *scene* (a collection of episodes), *episode* (a spoken and/or gestural interaction concluding with a control-flow decision), *do* (a gesture), *speak* (an utterance, specified via a string that can be output through a voice synthesizer, with an optional file name for a recorded version of the utterance), *pause* (for some number of seconds or fractions of seconds), *decide* (a conditional construct based on speech or gesture input from the human, go (to an episode), and *nextscene*. Table 1 presents the tags, their parameters, and their functions.

Table 1. Name, parameters, and functions of the tags

Tag	Parameters	Function
scene	scene name	Name of the file. Corresponds to a 3D environment in Unity with the same name
episode	episode name	Name of the episode. Episode tags can contain any tag except scene
do	emotion, hands, locomotion and upper body	These tags specify the agent's animations. You can specify an animation for the face (emotion), hands, lower body (locomotion) and upper body
speak	wav file and speech string	Contains either a.wav file name as a string, and/or the text of what the character will say
decide	speech id or gesture id	Makes a dialog branching decision based on user's speech or gesture input
go	target episode	jumps to a new episode upon the conclusion of the current one or after a decide tag conditions have been met
pause	seconds	Pauses the interaction. Useful to avoid further input while the agent performs a lengthy animation.
createrule	grammar rule name	Sets the name of a grammar rule.
tag	grammar tag	Serves as the name of a grammar slot to which several items are mapped to. Each grammar rule can have any number of tags
items	grammar items	Items are the words users
ruleCall	grammar rule name	Calls a previously defined grammar. After tags and items were provided, grammars can be reused in any section through rule calls
nextscene	scene name	The name of the new scene to be loaded

(There are also memory tags, such as store and recall, that connect to an SWI-Prolog back-end so that enables the ECA to maintain a dynamic dialog model.)

The scene interpreter, implemented in C# for convenient integration with other modules of the ECA system, parses the XML script, creates scene, episode, and action objects corresponding to the script's tagged elements, and sends the objects to the ECA's run-time system for execution.

Our approach contrasts with that of some other standards for specifying ECAs and their interactions. In particular, our system differs from the SAIBA framework (Zwiers et al. 2011) so that our ECAs can more easily integrate cognitive functions across what would be separate modules in a SAIBA-compliant system. SAIBA's strong modularity may not be appropriate for interaction that is fine-grained or requires rich contextual information (Lee et al. 2008). Similarly, the strong theoretical frameworks of BML (Kopp et al. 2006) and FML (Heylen et al. 2008) tend to impose cognitive and operational models that may be over-detailed and limiting for ECAs intended to explore the frontiers of human-agent interaction in practical terms. In a sense, architectures such as BML and FML handle the interaction from the agent's point of view. In our case, we need additional control over the environment and its navigation for actions by both the user and the agent.

Each scene is contained in a file. Inside each scene there can be any number of episodes. Each episode can contain tags that enable the agent to speak, listen, decide, act, and then jump to other episodes. In addition, each scene corresponds to a physical, navigable 3D representation of the environment. By changing scenes, one can change the scenery—and even the agent, if required.

Inputs from humans to the ECA system come through speech and gesture. The author's parameters for decide tag determine how the system processes these inputs and chooses the next episode in the scene. In Sects. 3 and 4, respectively, we discuss the handling of speech output and input, and gesture output.

3 Speech Handling

In this section we focus on the speech modules, and in particular on the *speak* and *decide* speech tags. We begin with the *speak* tag, which specifies the ECA's spoken language output. To illustrate the scripting language's handling of speech, Fig. 1 presents an abridged episode from an early scene in the "Survival on Jungle Island" game. In this episode, as in all others, *speak* tags contain both a written and optionally an audio version of the dialog specified; for example, in the script below the file decideu1.wav is a recorded version of the utterance. Because authors can use text strings without speech files, they can quickly prototype and test scenes using synthetic speech before committing to recordings. When executing behaviors indicated with a *speak* tag, the system uses the audio file name to find the audio for playback and has the agent lip-sync to the audio through a dialog tree, implemented as a separate module.

The *decide* tag signals a branching condition, which determines which will be the next episode. The agent asks a question and provides two or more possible answers. When the speech-recognition module recognizes one of the answers with high confidence, the system branches to a named episode, which is a parameter for the *go* tag.

```
<episode "splitpaths5">
    <do emotion "Neutral" hands "Idle" Locomotion "Default" upperBody "Thinking"
      isSpecial "false" special "none">
    <speak "splitpaths5u1.wav" "So, where to now? Should we take the grassy path
      which leads to the jungle or take the rocky path which leads to the
      mountains?">
    <pause "1">
    <decide speech>
        <createRule "paths">
            <go "grassy"> </go>
                <tag "grass">
                    <items "grassy path" "grassy" "lets take the grassy path"
                       "lets take the path no one has been to" "the grassy
                       path" "the jungle" "the grassy path, i am afraid of
                       the mountains" "the grassy path is safer" "the safest
                       path" "the left one" "to the left" the one on the
                       left">
                  </tag>
            <go "rocky"> </go>
                <tag "rock">
                    <items "rocky path" "rocky" "mountains" "the mountain
                       path" "the rocky road" "the right one" "the one on the
                       right" "let's seek altitude" "let's go to the moun-
                       tains" "we should take the rocky path" "the rocky path
                       to the mountain">
                </tag>
        </rule>
    </decide>
```

Fig. 1. A scene section using our markup-language. This script causes the agent to look thoughtful while asking a question (through a predefined audio file). It then pauses for a second to prevent the speech recognizer from listening to the playing audio file. Finally, a grammar is created inside a *decide* tag, which enables users to progress to the next episode or scene depending on their verbal expressions.

In the example in Fig. 1, "grassy" and "rocky" refer to other episodes of the scene, to which control will pass depending on the human's answer to the ECA's question.

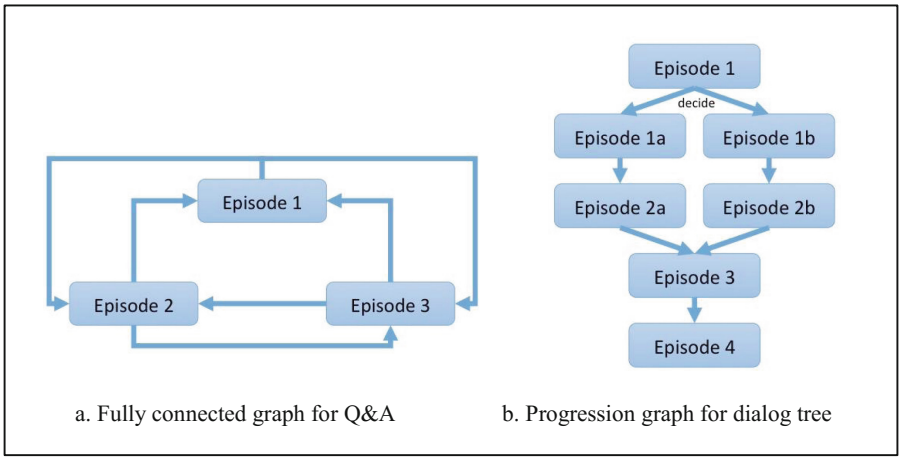
To improve reusability, the *decide* tag can transfer control to any episode in the scene, enabling authors to create dialogs that loop until certain criteria are met. The grammar itself contains every word or phrase that can be recognized by the system and is not affected by the current state of the conversations. A speech-recognition grammar is in turn composed of rules. Each rule defines what words or phrases should be recognized during a particular dialog state, which is what we define as episode. As the control flow swaps the episode, the rule changes as well to enable a different set of available words for recognition. The idea behind this rule-based grammar, in which the rules act as speech-recognition separators for episodes, is to enable only a small subset of the whole grammar to be active at any given time. This in turn helps improve the recognition by not engaging in grammar over-coverage.

By creating a rule, an author can refer to a speech-recognition grammar in future episodes without the need to declare the grammar again. In other words, if a rule has been declared, it can be called and reused again from the script by just specifying the name. If the rule is being used for the first time, its name, tags, and items must be declared. The only requirement is that the grammar has been used at least once before being reused. This is the function of the *createrule* tag. Figure 2 shows a rule initialized

from the script and a demonstration of how to reuse it, and Fig. 3 presents the standard grammar xml file that is created based on the script declaration.

```
<<scene "beach2">
    <episode "beach2c">
        <do emotion "Neutral" hands "Idle" Locomotion "StandUp" upperBody
          "Pointing" isSpecial "false" special "none">
        <speak "beach2au2.wav" "Here, let me help you up. Grab my hand.">
        <decide gesture>
            <go "trapped"> "Grab Hand" </go>
        </decide>
    </episode>
    <episode "trapped">
    . . .
    </episode>
</scene>
```

Fig. 2. In this scene the agent uses a combination of animations to extend its hand to the user. The agent then prompts the user to grab her hand through a decide tag. With only one option, it is a required gesture to progress through the interaction. After the user performs a "Grab Hand" the control flow decides on the next episode to execute. At the end, the scene changes. On the virtual side, a new environment is loaded.



a. Fully connected graph for Q&A b. Progression graph for dialog tree

Fig. 3. Graphs of episodes and transitions showing different styles of dialog paths

4 Gesture Handling

In this section we explain the implementation of the tags for performing and recognizing actions. In the example presented in Fig. 2, the interaction begins in scene "beach2" with the episode "start." The *do* tag accesses the animation tree to produce a blend of animations to produce the intended motion for the agent (Gris, Rivera & Novick 2015). The system uses a layered approach to animation that enables

developers to specify the agent's actions with enough control to represent the desired situation and emotion but with enough abstraction as to avoid getting lost in detailed descriptions of animations. Movements of the body are specified in layers for locomotion, emotion, upper-body, and hands.

The episode in Fig. 2 uses a *locomotion* layer, which enables the agent to walk around, move to the next waypoint, sit, crouch, crawl, or do any other movement that requires character displacement across the scene or lower body animations that change the current stance. In the example, the agent starts the scene by moving to a standing position, which is a valid transition because the default starting point for this particular scene was specified as crouching, even though agents generally start scenes in a standing position.

The *emotion* layer controls the configuration of the agent's face. All transitions are weighted and transitioned gradually between emotions; the transitions are handled by the system's additional animation-tree modules.

The *hands* parameter indicates the position or shape of the fingers. This is used for special circumstances such as pointing or making representational hand gestures.

Finally, the *upperBody* parameter specifies the action of the torso and arms. In the design of our ECA system and the mark-up language, we sought to avoid unnecessary or overspecialized animations. In the case of the scene in Fig. 2, the agent is extending its hand towards the user to help him or her cross a gap in the virtual scenery. The upper-body parameter is pointing because this enables reuse of another animation. A pointing animation includes an outstretched arm; without the hand performing a pointing gesture, too, the pointing gesture is just an extended arm with an open hand. This is one of the ways in which work is reused to create new interactions without additional low-level development work.

All parameters from these tags are parsed and stored to handle new agent gestures, providing flexibility and expandability of gesture libraries without having to delve too deeply in the implementation to access them. These animation libraries are created by combining multiple exchangeable parameters for separate body-part layers. After parsing, these gestures become available as they were specified, in an ordered sequence when called by the episode in control of the dialog state, however, developers can incorporate (outside of the markup language declaration) conditions or variables that affect the agent's gesture behavior depending on the runtime observations of the interaction. For example, a developer can specify within an episode a gesture containing surprise for the upper body animation layer, and happiness for the facial animation layer. If, however, this normally happy moment is obscured by a series of user transgressions that the developer kept track of throughout the interaction, the "do" tag queue can be accessed at runtime and animations can be overwritten. This provides additional control over the otherwise static declaration, although it requires an additional abstraction layer between the rendering application and the xml interpreter that we do not provide.

For gestures, the *decide* tag connects to the ECA system's gesture-recognition module. In episode in Fig. 2, the agent waits for the user to extend his or her hand towards the screen. The gesture is then captured and processed by a Kinect and our gesture-recognition software. Similar to the way the decide tag works for speech, this list can contain several gestures leading to different episodes, depending on the gesture

performed by the human. Depending on the gesture that the user performs, the *decide* tag will transfer control to the appropriate episode. This is also useful to create activities that require a gesture response to progress. Gestures are specific to the application; in the case of our most recent ongoing project, they represent survival activities, such as lighting a fire, spear fishing, and signaling for help.

5 Conclusion

We developed an XML-based scene markup language that supports a variety of features for speech, gesture, memory, and scene handling in human-ECA interaction. Our approach limits detail of gesture animation in favor of abstraction and reusability. We focused on making the markup language extendable and adaptable for use as mid-layer software for agents using different architectures.

The scene markup language enables authors of scripts to develop dialog paths that can vary depending on the nature of the application. In a Q&A application, the dialog path will revisit states multiple times and in any order, as shown in Fig. 3a, as a fully connected graph. In a storytelling application, the dialog path will follow a decision-based progression, as shown in Fig. 3b, as a dialog tree. The markup language enables both forms to be created by adjusting parameters and rearranging the episode order.

In the markup language, tags are flexible and new tags can be added; however, changes made to the structure will require the users to update their scripts. Our language enforces consistency but does not provide it automatically.

We have successfully tested our mark-up language in two different projects. First, we wrote scripts for eight scenes that take advantage of scene transition, episode transition, gesture management, and dialog management. Second, we updated our previous agents to our new architecture. Where our previous agents took months to develop, the updated versions were recreated from scratch in a few hours, including additional animation, gesture and lip-sync features.

A major advantage of this approach is the relative ease of authoring scenes. Rather than having to write procedural code, which is typically hard to understand, modify, and debug, scene authors can create scenes declaratively and more easily reuse or adapt episodes. Undergraduate computer science students in a course on innovation in technology were able write scripts efficiently, producing a dozen scenes in which they were able to express creativity in the human-agent interaction rather than be concerned with implementation.

Our approach has limitations of design and implementation. In terms of design, our approach is fundamentally limited by its orientation toward scene-driven interaction. For human-ECA systems that are more about spontaneous, less-constrained interaction than episodic storytelling, an FML/BML approach might be more appropriate. In terms of implementation, we found that rewriting scripts to conform to updates in the specification of the markup language or changes to our file-naming conventions turned out to be tedious. Improvements to the system would include a tool for automatic updating of scripts to conform to changes in the scripting specification and a tool to update filename changes when you update the script. For example, if the naming

convention for audio files changes, the tool would spare the author from having to update manually all names for both the scripts and the files. As another example, we plan to update the tags associated with the grammar rules so that the tag names are createRule and callRule (instead of ruleCall) and so that these tags close with/createRule and/callRule (instead of/rule, which is an artifact from system's iterative development).

A second implementation improvement involves extending the availability of gesture interpretation. In addition to detecting gestures specified in the decide tags, our system also tracks the human's conversational gestures throughout the interaction session outside the decide tags. These include gestures such as crossed arms, normal stance, hand(s) on hip, hand(s) on face, and many others. Currently, these gestures are recognized through a background process that produces a gesture-annotation file for research of human-ECA interaction. It might be helpful to authors to integrate this annotation functionality to the markup language declarations so that they could access the human's gestures in real time.

A third implementation improvement would provide a higher-level authoring tool. Currently, authors create scene scripts by writing XML code directly, producing files similar to those in Figs. 1 and 2. Our plans for future work include developing a WYSWYG authoring system for the scripts that would generate the XML files automatically. This would enable non-technical authors to develop new scene scripts. As it is, we use the current version of the system for generating new scenes for the "Survival on Jungle Island" game and will be using the system to create our next-generation applications.

References

Anderson, T., Galley, S.: The History of Zork. The New Zork Times, New York (1985)

Crowther, W., Woods, D., Black, K.: Colossal cave adventure. Computer Game. Intellect Books, Bristol (1976)

Rayon, A., Gris, I., Novick, D., Camacho, A., Rivera, D.A., Gutierrez, M.: Recorded speech, virtual environments, and the effectiveness of embodied conversational agents. In: Bickmore, T., Marsella, S., Sidner, C. (eds.) IVA 2014. LNCS, vol. 8637, pp. 182–185. Springer, Heidelberg (2014)

Gris, I., Novick, D., Rivera, D.A., and Gutierrez, M.: UTEP's AGENT architecture. In: IVA 2014 Workshop on Architectures and Standards for IVAs, Intelligent Virtual Agents (2014), Boston, August 2014

Gris, I., Rivera, D.A., Novick, D.: Animation guidelines for believable embodied conversational agent gestures In: HCII 2015, Seattle, 2–7 August 2015 (in press)

Heylen, D., Kopp, S., Marsella, S.C., Pelachaud, C., Vilhjálmsson, H.H.: The next step towards a function markup language. In: Prendinger, H., Lester, J.C., Ishizuka, M. (eds.) IVA 2008. LNCS (LNAI), vol. 5208, pp. 270–280. Springer, Heidelberg (2008)

Kopp, S., Krenn, B., Marsella, S.C., Marshall, A.N., Pelachaud, C., Pirker, H., Thórisson, K.R., Vilhjálmsson, H.H.: Towards a common framework for multimodal generation: the behavior markup language. In: Gratch, J., Young, M., Aylett, R.S., Ballin, D., Olivier, P. (eds.) IVA 2006. LNCS (LNAI), vol. 4133, pp. 205–217. Springer, Heidelberg (2006)

Lee, J., DeVault, D., Marsella, S., Traum, D.: Thoughts on FML: behavior generation in the virtual human communication architecture. In: AAMAS (2008)

Novick, D., Gris, I.: Building rapport between human and eca: a pilot study. In: Kurosu, M. (ed.) HCI 2014, Part II. LNCS, vol. 8511, pp. 472–480. Springer, Heidelberg (2014)

Zwiers, J., van Welbergen, H., Reidsma, D.: Continuous interaction within the SAIBA framework. In: Vilhjálmsson, H.H., Kopp, S., Marsella, S., Thórisson, K.R. (eds.) IVA 2011. LNCS, vol. 6895, pp. 324–330. Springer, Heidelberg (2011)