A Novel Realizer of Conversational Behavior for Affective and Personalized Human Machine Interaction - EVA U-Realizer -

IZIDOR MLAKAR[1], ZDRAVKO KAČIČ[1], MATEJ BORKO[2], MATEJ ROJC[1]
[1]Faculty of Electrical Engineering and Computer Science, University of Maribor
[2]A1 Slovenija
SLOVENIA
izidor.mlakar@um.si, kacic@um.si, matej.rojc@um.si, Matej.Borko@A1.si

Abstract: - In order to engage with user on more personal level, natural human-machine interaction is starting to virtual or even physical entities resembling human collocutors. Namely, through human-like entities the designed multimodal interaction models, try to adapt to user's context and to facilitate context of the conversational situation. The designed multimodal processes tend to follow the ''rules'' and ques as implanted during face-face interaction among humans. In this way, the integration and exploitation of embodied conversational agents (ECA) for human-like interaction seems only natural. The ECA's artificial body and articulation capabilities are already close to those found on real humans. From skin, face, hands, and body posture, these virtual entities tend to look and behave as realistically as possible. Furthermore, ECAs tend to imitate as many features of human face-face dialogs as possible, and integrate them into interaction as synchronized as possible. One of the essential functions in face-to-face interaction is the ability to reproduce synchronized verbal and co-verbal signals coupled into conversational behavior. Other signals, such as: social cues, attitude (emotions), personality, eye-blinks, and spontaneous head movements are equally important, and have to be blended into multimodal expressions. However, designing a realistic entity that acts realistically like humans do, is a daunting task. Modern 3D environments and 3D modeling tools, such as: Maya, Daz3D, Blender, Panda3D and Unity, have opened up a completely new possibilities to design virtual entities, which appear almost (if not completely) like real-life persons. However, the modern 3D technology mostly covers the design and deployment part of such realism, while the realism of the behavior itself and its diversity and dynamic nature are generally not the focus of 3D modeling frameworks. Namely, most of the animation prepared in 3D frameworks is planned, designed in advance, and over well-written and well-designed situations. Therefore, it integrates limited diversity as well as limited capacity to adapt to a new set of parameters. As a result, 3D frameworks have a limited capacity to handle highly dynamic and interchangeable contexts present in human interaction. In this paper we, therefore, outline EVA U-Realizer, the second generation of proprietary behavior realization module. The goal of the realizer is to integrate the diversity, responsiveness, and adaptiveness that is required for the facilitation of conversational responses with animation capacities and realism provided by 3D modeling frameworks. As a result, the presented novel realizer is built over Unity 3D game engine's core. The proposed realizer considerably improves the capabilities of the animation engine itself, by providing interpreter and executor for incorporating dynamic and real-time generated conversational artefacts, similar to those found in conversations among humans.

Key-Words: - embodied conversational agents, personalized interaction, co-verbal behavior, behavior realizer, animation, virtual reality, mixed reality, multimodal interaction

1 Introduction

In the last few years we can observe that conversation is becoming a key model in human machine interaction [1]. One of the key challenges in the modern human-machine interaction (HMI) is the generation of more natural output. Namely, the emergence of several new devices that are already capable to support multi-media conversational agent (CA) technology is gaining traction in user interfaces [24]. The CAs have, therefore, actually become an indispensable tool in everyday scenarios for advanced HMI. Namely, from Apple and Microsoft, to Amazon, Google and Facebook all have adapted their own variations of CAs. The CAs range from chat-bots and 2D cartoon-like realizations of talking heads [2,3,4,5], which are used primary as the human-like representation for visual speech synthesis, to fully articulated embodied conversational agents (ECA's) that are able to perform interaction in various concepts including sign language [6,25], storytelling [8], companions and pedagogical agents [26,27], and as virtual hosts for various user interfaces [7,13,28]. In general, the embodied Conversational Agents (ECAs) are, nowadays, the best choice for development of natural and human-like machine interfaces. Recent studies in the field of face-to-face conversation show that the proper synchronization of verbal and co-verbal signals represents the key element for recreating truly natural interaction (gestures and expressions)[9]. Namely, the co-verbal behavior represents a major source of discourse cohesion. It regulates communicative relationships and may support, or even replace the verbal counterparts [10]. Thus, the co-verbal behavior effectively retains semantics of the information and adds a certain degree of clarity in the discourse. Further, it clarifies collocutors communicative goal, and reflects psycho-social nature of given information, through social and psychological responses, attitudes, and personality. An ideal user interface would, therefore, integrate most (if not all) artefacts that humans use while talking to each other. Human interaction also implies that classical direction-based or task-oriented functionality should be replaced with the conversation oriented tactics, in order to make the dialogue more natural and more personalized [30, 31]. Overall, ECAs (human like virtual characters with embodiment) foster the capacity to resemble and to foster the natural way of reacting and interacting, and they can select and display appropriate non-verbal cues.

However, natural multimodal interaction is much more than speech accompanied with the repetitive movements of limbs and face. Natural interaction entails multiple behavior variations that are correlated in a dynamic and highly unpredictable settings [11]. Furthermore, natural conversational behavior also incorporates various social and interpersonal signals, in order to ‘color’ the final outcome. Thus, the virtual entity must behave like human in any situation. Further, it must also have the capacity to dynamically adapt itself to the social and other situational contexts [1,12,27,32]. The design of ECAs, therefore, represents a complex and still a daunting task. Nowadays, game engines, such as: Unity 3D[1], Panda 3D[2], Irrlicht Engine[3], Ogre3D[4], are also must-have tools for developing 3D Virtual Reality environments in the field of embodied conversational agents. Namely, in combination with various 3D modeling tools (e.g. Maya, Daz3D, Blender), the production of highly realistic humans is then much easier and affordable. Nevertheless, when a virtual character is to be represented as a viable and realistic imitation of a human, it must also facilitate human like behavior. Body motion, postures, and other artefacts of non-verbal behavior must be aligned with various contexts and aspects of the conversation situation and everyday life. In the field of embodied conversational agents, in general the *intent planner* and *behavior generator* are responsible to produce human like behavior. The behavior is composed of speech acts, communicative intents, and non-verbal signals (gestures and expressions) that are synchronized into common expressive reaction. The task of the *behaviour realizer* is the proper realization of the behaviour regarding time, on the targeted virtual entity [13]. The game engines and 3D modelling frameworks provide a perfect development environment for design and deployment of realistic virtual entities, and their highly realistic animation. However, they fail to satisfy several believability parameters of conversational behaviour, such as: diversity and multimodal planning, situational awareness, synthesis of verbal content, synchronization, etc. E.g. if the human eyes are exposed to the same (or quite similar) sequences of movement, then in time, they will start to appear less and less natural. Thus, the diversity and the ability to use several variations of visual cues in the same context, as well as a variation of a visual cue in various contexts, is one of the key things in order to achieve truly natural interaction. The coupling of behaviour *planners* and *generators* (e.g. behaviour specification tools) with the animation components is, therefore, only natural. Namely, the behaviour specification components on one hand, bring diversity and synchronization with advanced interaction models and vast repositories of non-verbal cues. The modern animation engines on the other hand, take care of the realism of the agents and the scenes, as well as the realism of the animated movement [34, 35].

In this paper we represent a framework for rapid design and development of embodied conversational agents that are capable of realizing dynamic and complex conversational behaviour. We follow the modular concept of separating *behaviour specification* and *realization* into two independent processes. The EVAScript acts as interlink between the two [14]. And EVA behaviour generator

[1] https://unity3d.com/

[2] https://www.panda3d.org/

[3] http://irrlicht.sourceforge.net/

[4] http://www.ogre3d.org/

specifies the behaviour, while the EVA Realizer animates it [15].

In this paper we represent our latest effort towards exploitation of 3D modelling framework for the deployment of more natural HMI. Namely, we integrated Unity 3D game engine's powerful functionalities into proprietary ECAs behaviour generation framework, by integrating them as an additional instance of the EVA Realizer. Already available EVA realizer ran over Panda 3D core. In contrast to Panda 3D technology, we can achieve far greater realism of the environment and virtual characters, when using Unity. This is mainly due to far more controllable and event oriented virtual environment that we are able to establish by using the Unity environment. However, in terms of procedural animation and integration with EVA concepts, the native mechanism of animation and control in Unity were less compatible with our concept of co-verbal events. Thus, this required additional research effort, in order to develop the animation engine within the EVA framework, and to exploit its features for behaviour realization.

To sum up, this paper is structured as follows. In the next section we outline the motivation behind the research in the field of the conversational behaviour realization. Thus, the next section overviews relevant research efforts, and points out the key advantages and benefits of the Unity engine. We then continue with the outline of the overall architecture of the ECA framework, and how it is designed and used in our studies in HMI as a whole. Next, we focus on the development of behaviour realization engine in particular. We outline and describe its architecture, implementation, and the envisaged event-oriented animation model. The paper is concluded with achieved visualizations of co-verbal behaviour on female and male agents, followed by a discussion and final remarks.

2 Related works

Conversational characters are becoming increasingly more realistic and human like. However, human eyes see every detail and even a small discrepancy in outlook or movement may already appear highly unnatural and synthetic. Furthermore, if users are exposed to a similar visual stimulus for longer periods, the movement will over time, start to appear less and less natural. Thus, the diversity of the repository of non-verbal stimuli and the overall animation of conversational behaviour (including interrupts and changes) are equally important for the appearance and the actual synchronization. In the field of conversational agents, the animation part and handling of the virtual environment is usually left to a (rudimentary) *realizer* component. The most spread are those *realizers*, which handle behaviour events specified in BML. Among BML realizers, Unity 3D has been chosen as the preferred animation engine. Nevertheless, the Virtual Human Toolkit (VHTK) [16] uses SmartBody [17] as the underlying realization and animation engine. Further, E-VOX [33] is emotionally enhanced semantic ECA, which supports the realization of expressive facial gestures based on an emotional model inspired by ALMA. This allows the ECA to take into account features needed for social interaction. Greta [18] also supports the realization of expressive conversational behaviour based on MPEG4 BAP-FAP layers. In this case the platform facilitates Ogre[5] game engine for the animation. There is also an ongoing effort to deploy Greta over Unity game engine[6]. Further, Elckerlyc [19] is a BML *realizer* for generating multimodal verbal and nonverbal behaviour for Virtual Humans (VHs). It is a model-based platform for the specification and animation of synchronised multimodal responsive animated agents. EMBR [20] is also a BML *realizer* that offers a high degree of animation control via the EMBRScript language that is used as interlink between the behaviour generator, and the animation engine. EMBR is based on Panda 3D[2] as the underlying animation engine. An ASAP Realizer-Unity3D Bridge [21] also represents BML realizer facilitating Unity game engine as the animation and rendering engine. This system combines the benefits of a modern game engine and a modern BML *realizer*.

Similarly, as these systems, we tried to exploit capabilities of modern game engines in development of novel behaviour generation engines. The EVA *realizer* presented in [22], is already capable of animating high-quality conversational behaviour consisted of gestures, facial expressions, lip-sync, gaze and head movement, and posture. The *realizer* facilitates procedural animation delivered via EVAScript language, synthesis of subconscious behaviour in form of spontaneous and expressive eye and head movements, as well as targeted animation in form of *LookAt*, *PointAt* and *Follow* commands. This version of the *realizer* presented in [21], has been implemented based on Panda 3D environment[2], while novel development

[5] Ogre3D: http://www.ogre3d.org/

[6] Greta in Unity: https://trac.telecom-paristech.fr/trac/project/greta/wiki/GretaUnity

achievements regarding EVA *realizer* are presented in this work, where new possibilities of Unity game engine are exploited.

To sum up, modern behaviour *realizers* have the capacity to support several parameters of believability of conversational behaviour, such as: diversity and multimodal planning, situational awareness, synthesis of verbal content, synchronization, etc. They, however, lack in production capacities in terms of agent and environment, as well as the visual realism and integration into various interfaces. Game engines on the other hand are powerful tools for rapid and high quality design and rendering of virtual human-like entities including ECAs. They enable the design and delivery of beautiful and highly realistic graphics, and the efficient handling of hardware resources. The game engines, however, lack in the capacity to dynamically adapt movement artefacts to various conversational, social, and other situational contexts. Thus, game engines provide only limited (if any) mechanisms to generate and deliver the diverse set of co-verbal artefacts, which are synchronized over multiple channels and adapted with the context of situation.

EVA U-Realizer presented in the paper significantly improves proprietary behaviour realization module presented in [22]. Architecturally, both realizers are quite similar in the design. Nevertheless, important advantages of the Unity-based implementation over the Panda 3D – based implementation are the following:

- Integrated animation scheduling and frame by frame control, with the capacity to control the scheduled animation and even animation already being executed. In contrast, the Panda 3D *realizer* implements stop and interrupt procedures to handle new events. Further, the scheduling is quite rudimentary, in form of FIFO ques that hold the pre-processed animation sequences to be realized. The scheduling in Unity-3D *realizer* incorporates associative arrays and allows for a) replacement of sequences, b) replacement of parts of sequences, and c) changes made to the scheduled sequences and sequences already being played-out.
- Direct support for inverse kinematics (IK) supporting targeted animation (e.g. *LookAtObject* and *FollowObject* behaviour). Panda 3D *realizer* only partially supports IK via external libs and implanted *LookAt* and *Follow* behaviours, via transformation of 3D position to the rotations of neck joint and arm joint chain (online estimation).
- Specification of customized interpolation curves for more natural smoothing of the animation. The Panda 3D *realizer* supports Linear, EaseIn, EasOut, and EasInOut interpolation, where curves are static and cannot be managed. In Unity 3D *realizer* we support virtually any kind of mathematical function used to calculate the 'in-between' configuration of movement controller in relation to the current time-frame.
- Integrated scene and animation editor with the ability to visually set-up the environment parameters, as well as to design agents' posture and gestures. Off course, 3D modelling tools, such as: Maya 3D and Daz 3D, are still supported and being used. The Unity 3D *realizer* actually supports direct import of Autodesk's FBX. In turn, this significantly reduces the complexity of the integration of 3D scene designed in 3D modelling environment, and its facilitation in the interactive environment (e.g. ECA environment).
- Deployment to various mobile and stationary platforms. Thus, richer support for various communicative contexts in various environments. Panda 3D *realizer* only partially supports this option via web clients.

However, there are also some cons to the Unity 3D *realizer*. For instance, Unity3D is proprietary and closed source game engine. In contrast, the Panda 3D game engine is free and open source. Unity3D is also quite self-centred engine. Thus, it uses very unique approach for doing 'things'. Most of the knowledge and mechanisms are not directly transferable to other engines. Finally, for efficient memory handling, classes have to be implemented in order to manage objects in memory (e.g. object pooling).

3 The EVA Framework Architecture

The general concepts of the EVA framework architecture are in line with the related research achievements in the field. Thus, we follow the principle of the component modularization, similar to the one outlined in the SAIBA architecture [23]. Architecture of the EVA Framework and the EVA *realizer* in particular is outlined in Fig. 1, where we outline the overall framework that we have envisaged and deployed in order to facilitate more natural human machine interaction, as can be delivered by using ECA technology. It also specifies those particular components that have been designed and developed for the transformation of the co-verbal behavior into speech synchronized animations performed by an embodied conversational agent, named EVA.

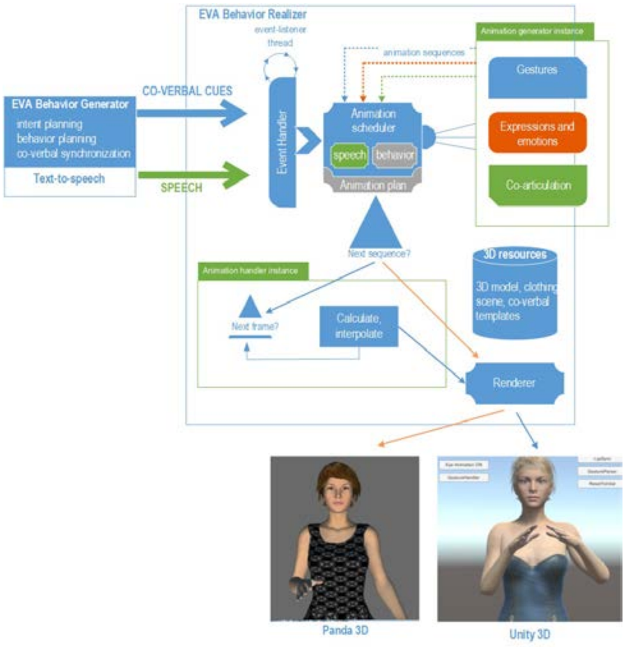


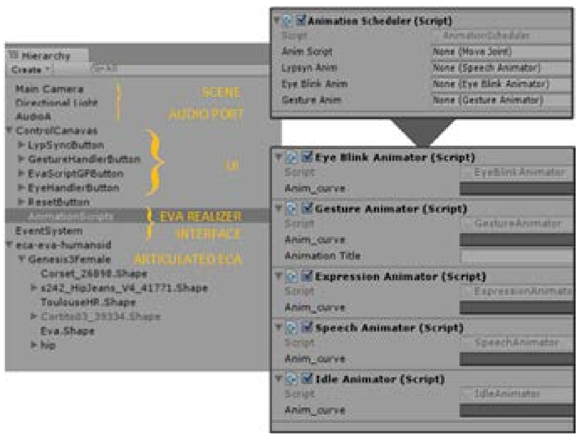
Figure 1: The framework for the delivery of natural human-machine interaction - EVA Framework

The architecture of the EVA Framework, as outlined in Fig. 1, is highly modular and includes an 3rdParty game engine, which is responsible for generating the conversational behavior in form of synchronized co-verbal (e.g. gestures and expressions) and verbal acts (e.g. speech). Within the EVA framework, the component is named EVA *Behavior generator* [15]. This is actually an omni-comprehensive TTS engine, which generates the synchronized conversational behavior in form of conversational events. Each event consists of non-verbal events described in EVA Script, and are aligned with the synthesized verbal counterparts.

The second component is the EVA *Behavior Realizer*, which is also the focus of this paper. It realizes the generated synchronized behavior and represents it to the user. In order to do that, the EVA *Behavior Realizer* transforms conversational events into their physical representations. This is achieved by applying the various co-verbal features described in the co-verbal events to the 3D resources available in the *renderer*. In addition to scene artefacts and camera, the more important artefacts are the exposed controlled objects, which are used to manipulate the embodiment of the virtual character (e.g. its movement controllers). Upon completion of the animation sequence (and the whole co-verbal act), a feedback in form of a conversational context, is also sent back to the behavior generator, or any other component requiring it (e.g. dialog manager). As part of the EVA framework, so far we have designed and deployed two EVA *realizers*, one is based on Panda 3D game engine, and the other one, represented in this paper, is based on Unity 3D engine. The main components of both EVA *realizers* are: *Event Handler*, *Animation Generator*, *Animation Scheduler* and *Animation Handler*. The *Event Handler* intercepts and handles the conversational events. It parses event stream, and checks for the event type and the event priority. The *Animation Generator* then transforms the conversational behavior into animation sequences. As part of this process, the *Animation Generator* applies temporal and spatial constraints adjusted to agents' articulated body. The *Animation Scheduler* then inserts the generated animation sequences into the execution plan. It handles the animation graph and feeds them to the rendering engines accordingly. Finally, after the realization of each animation sequence is completed, the *Event Handler* signals its status (conversational context) to the behavior generator, and possibly also to the dialog handlers. Similarly, after the full realization of the behavior que, a change in conversational context event is raised and the generation of inactive (rest) behavior is triggered.

3.1 The EVA U-Realizer: Architecture and Implementation

In order to integrate and fully facilitate all aspects of the conversational behavior supported by EVA framework, the design and development of the EVA U-Realizer follows the approach presented in [14]. The basic architecture and implementation of the EVA U-Realizer is outlined in Fig. 2.



```
Main Camera
Directional Light
AudioA
▼ ControlCanavas
  ► LypSyncButton
  ► GestureHandlerButton
  ► EvaScriptGPButton
  ► EyeHandlerButton
  ► ResetButton
  AnimationScripts
EventSystem
▼ eca-eva-humanoid
  ▼ Genesis3Female
      Corset_26898.Shape
    ► s242_HipJeans_V4_41771.Shape
      ToulouseHR.Shape
    ► Cartited3_39334.Shape
      Eva.Shape
    ► hip
```

```
Animation Scheduler (Script)
  Script          AnimationScheduler
  Anim Script     None (Move Joint)
  Lypsyn Anim     None (Speech Animator)
  Eye Blink Anim  None (Eye Blink Animator)
  Gesture Anim    None (Gesture Animator)

Eye Blink Animator (Script)
  Script          EyeBlinkAnimator
  Anim_curve
Gesture Animator (Script)
  Script          GestureAnimator
  Anim_curve
  Animation Title
Expression Animator (Script)
  Script          ExpressionAnimato
  Anim_curve
Speech Animator (Script)
  Script          SpeechAnimator
  Anim_curve
Idle Animator (Script)
  Script          IdleAnimator
  Anim_curve
```

Figure 2: The EVA U-Realizer architecture

The 3D resources outlined in Fig. 1, are represented by SCENE and ARTICULATED ECA components, as outlined in Fig. 2. The ARTICULATED ECA component maintains all 3D resources designed for each separated virtual agent and the overall scenes. The 3D objects are either physical animation objects (3D resources in Fig.1), such as: lights, objects in environment, cameras, interpolation curves and the articulated virtual characters, meshes, skeletal structure; or C# algorithms (scripts) implementing targeted functionality encapsulated into C# classes (e.g. animation scripts and event system in Fig. 1). The C# classes extend the Unity's native *MonoBehaviour* class.

The physical 3D resources are actually those 'tools', through which body-part based animation and co-verbal behavior are implemented. In addition to gestures, gaze, facial expressions and lip-sync, the agent can also engage with various artefacts in the environment. This implies that a proper implementation of an articulated agent and its embodiment in addition to interactive objects in the environment are exposed to the game engine. For the design and the implementation of the agent, the Unity directly supports most of the popular 3D formats from obj, collada, and fbx to even direct import of Maya scene files (.mb). Furthermore, the interaction between agent's embodiment and objects in the environment can also be inherited from the imported 3D scene.

As already mentioned, the EVA U-Realizer is implemented as a series of algorithms and scripts extending Unity's *MonoBehaviour* class. *MonoBehaviour* class is the base class of all scripts that can be attached to the 'game' objects, and can manipulate objects in the scene. Furthermore, each class of type *MonoBehaviour*, automatically implements event functions. These functions are inserted into the programs lifecycle and executed in a predetermined order[7]. Among more important event functions are the following:

a) *Start*, which handles all the initialization procedures, and is called just once for a given script.
b) *On* (MouseXXX), which handles scene events natively supported by the engine.
c) *Update*, which handles 'game' logic at each frame. Co-routine updates are run after the Update function returns. This function performs the key mechanism for the implementation of frame-by-frame operations of the EVA U-Realizer.
d) *OnDestroy*, which handles the decommissioning of an object. It actually specifies what happens, when an object is destroyed. This function is called after all frame updates for the last frame of the object's existence (the object might be destroyed in response to *Object.Destroy*, or at the closure of a scene).

Animation Scheduler is the central component in the EVA U-Realizer. The *Animation Scheduler* is represented by the *AnimationScheduler* script, which manages all other resources and threads. It loads 3D resources, builds the scene, and initiates all other sub components, such as: animation generators represented by *Animator* scripts, and the *Animation* and *Event* handlers.

The *Event Handler* is represented by *EventSystem* object, which extends Unity's native event handling framework (and the event oriented lifecycle of the application) by incorporating EVA conversational events and publish-subscribe (unsubscribe) mechanism for handling of particular animation streams and contextual events beyond the scope of 'mouse', 'keyboard', and similar input events supported internally by Unity. It is initiated by the *Animation Scheduler* in the beginning of the applications' life cycle. It implements all event-related function of the *MonoBehavior* class, through which it handles internal events. In addition to handling internal events, it also acts as a bridge between the EVA U-Realizer and the rest of the EVA Framework. For this functionality, the event handling framework is extended with publish-subscribe mechanism based on the .NET event handling and delegates' concepts[8]. In general, the *Event Handler* subscribes to listen to all events in the system. Thus, when a sub-component (e.g. animation handler, EVA event handler) raises an event, the *Event System* intercepts it. If it is relevant it processes it, and at the end posts a proper request to the *Animation Scheduler*. In addition to handling internal events (and the communication between all the components exposed in the virtual 3D space), the *Event System* also acts as a bridge between the EVA U-Realizer, and the rest of the EVA-Framework. Thus, it exposes the EVA U-Realizer to the rest of the "world". This is achieved via EVA *Event* handler sub-component that after some EVA event is received, triggers an internal event relevant for the *Event Handler*.

Next, the EVA U-Realizer implements several animation generators for planning idle/sub-conscious behavior, and three generators for interpreting the co-verbal behavior. The *Eye Blink*

[7]https://docs.unity3d.com/Manual/ExecutionOrder.html

[8]https://docs.microsoft.com/en-us/dotnet/standard/events/

Animator is a script used to generate sub-conscious eye-blinks. The *Idle Animator* is then a script that interprets and generates the behavior of the agent, when it is not performing any interactive function. Such behavior for instance, includes spontaneous head movement and facial gestures. The *Event Handler* treats all idle/subconscious generators as regular animators. This is archived by designing the idle/subconscious models, which are engaged on demand by the *Animation scheduler*, in order to trigger proper EVA co-verbal events, which are relevant to the *Event Handler*. When animation scheduler has no more co-verbal sequences to execute, it will start the idle/subconscious models and indirectly signal the *Event Handler* to deploy the idle/subconscious behavior. However, when an EVA event is received, the *Event Handler* will pause idle/subconscious models, remove the planned idle/subconscious behavior, and ensure for a smooth transition between the last frame of the idle/subconscious models sequence to the first frame of the co-verbal sequence.

Finally, the *Animation Scheduler* supports three independent animation streams: one for speech, one for facial expressions, and one for gestures. Each of the streams is handled by one of the *Animation Handler*'s components (e.g. Animators). The *Animation Handlers* transform EVAScript descriptions into sequences of agent's embodiment configurations, and into those time intervals as specified in the EVAScript language. Each of the *Animators* prioritizes over a separate segment of embodiment and co-verbal artefacts. Namely, it will have higher priority, when several *Animators* try to configure the same segment. For instance, *Speech Handler* targets primarily the visualization of phonemes (vizemes). Thus, it will have higher priority in comparison to the movement controllers in the lower facial region. However, it will share them with the *FacialExpression Handler*. In this way, both handlers are capable of controlling movement controllers in the lower facial region. However, when both try to configure them at the same time, the *Speech Handler* will provide the highest influence, what lies in line with the animation blending concept. Thus, e.g. in this way the agent can display a vizeme with a smile.

In general, the *Animator* represents the link between the rendering engine and the objects in the scene, including the articulated embodiment. Speech is visualized via the *Speech Animator*. Further, facial expressions are animated via the *Expression Animator* and *Gestures* (including posture), while head movements (including gaze) are handled via the *Gesture Animator*. In order to properly interpret EVAScript language, and to deploy animation blending, each *Animator* deploys an *Animation* generator. The *Animation Generator* applies the sequences of agent's embodiment configurations, as described within the particular co-verbal event, into an internal animation objects adapted to both the scene, and the targeted section of the agent's embodiment. Thus, the generator is responsible for: a) applying EVAScript descriptions onto designated parts of embodiment, b) to apply configurations for subconscious/idle behavior, and c) interaction with various concepts in environment, such as: Follow/LookAt object, Grab object, etc.

4 The Publish/Subscribe Model and Frame-by-Frame Operation

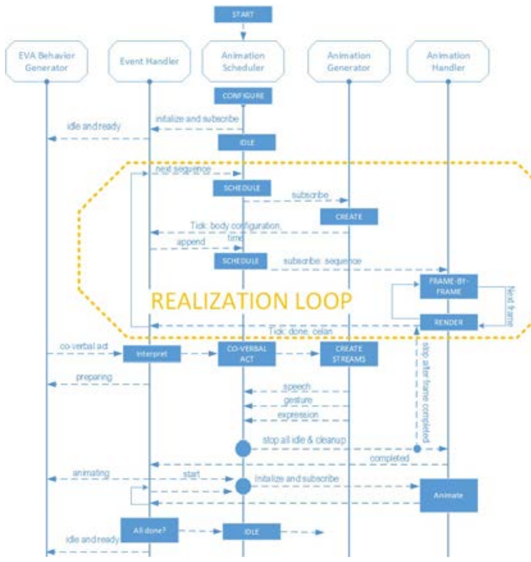


Figure 3: The event diagram of the event model.

As already mentioned, the communication between several processes within the realizer is implemented via event-oriented publish/subscribe model. Fig. 3 outlines the event diagram of the event model implanted for the EVA U-Realizer. In the architecture of the EVA U-Realizer, the model is managed by the *Event Handler* that is interconnected with the *Animation Scheduler*. Namely, the *Animation Scheduler* has to store the information about the virtual character (or characters) that is being controlled, and also the scene (including all controllable objects). Among more important data is the nature of agent's embodiment, which includes the composition of the skeleton and the face, as well as the initial/rest configurations of the embodiment and possible temporal and spatial constraints of the articulated body. These are loaded into realizer's environment upon initialization (e.g. configuration phase). Upon initialization, the *Animation Scheduler* initiates the *Event Handler* sub-component, and registers itself into the event system as the master. It also exposes all available *Animation Handlers* it can manage, along with available idle/subconscious behavior generators. All further functionality of the *Event Handler* is implemented within the scope of the unity's native *Update()* event-function.

After the initialization and configuration stage, the *Animation Scheduler* contains an empty schedule. Thus, it will trigger the realization of the idle/subconscious behavior. As outlined in Fig. 3, the *Animation Scheduler* firstly initializes all the necessary animation generators as intendent instances, and then starts the realization loop.

In the realization loop, the *Animation Scheduler* subscribes to each of the idle/subconscious behavior generators, and 'asks' them to generate an animation sequence. Each of the idle/subconscious behavior generators then implements its own algorithm over its own time-line, and controls a set of unique movement controllers. For instance, the *Eye-blink* generator controls facial controllers around eyes, including muscle movement implemented as blend shapes, and eye-lid movement implemented via 3D joints (bones). The *Animation Scheduler* starts the behavior generation by initiating the *Eye-blink* generator. When the eye-blink should occur, the *Eye-blink* generator raises an event and dispatches it to the *Event Handler,* and back to the scheduler, via *Event System*. And after the "eye-blink" event is completed, the *Animation Scheduler* asks for the next "eye-blink" sequence. The same concept is introduced for all subconscious/idle behavior handlers, as well as, when handling those co-verbal acts generated by the external systems. The proposed system all processes are triggered via events and all processes are completed via events. Thus, the execution is completely asynchronous and such processes are actually co-routines. Furthermore the asynchronous operation of co-routines ensures that no co-routine may interrupt the execution of another parallel co-routine. However, any co-routine may interact with another co-routine via the Event System. For instance, when *Animation Scheduler* (e.g. co-routine 1) requests for an "eye-blink (e.g. co-routine 2), it will continue with its operation until an announcement regarding animated sequences being prepared is intercepted. When in the meantime a co-verbal event is received, the scheduler is able to process and execute it (e.g. co-routine 3), and simply disregard or reschedule the animation of the idle/subconscious behavior. Namely, when some co-verbal event is marked as relevant, the *Animation Scheduler* inserts the complete sequence into its schedule.

In case of subconscious "eye-blinks", the *Animation Scheduler* will start appropriate *Animation Handler* and feed it with the behavior generated by the internal "eye-blink" generator. Similarly, the *Animation Handler* will also handle those co-verbal acts received as EVA events generated by external components. The difference is mostly in the structure of the animation sequence generated by the generator. Namely, the generators for idle/subconscious behavior will always generate a single sequence, while the next sequence will be generated only when the current sequence is completed and the algorithms behind the subconscious generator plan it. In case of co-verbal acts the co-verbal act already contains a complete behavior plan. Such a plan includes the manifestation of multiple embodiments, which are synchronized with speech. The temporal domain of the behavior must be retained. Thus, when the co-verbal act has been transformed into animation sequences, the *Animation Scheduler* will contain a series of sequences to be realized, as opposed to a single sequence, when animating eye-blinks. Furthermore, in terms of animation blending, the co-verbal acts always have the priority over subconscious behavior. The idle behavior (e.g. 'random' poses) is never executed, if some co-verbal acts exist.

During the execution of the realization loop, the *Animation Scheduler* picks up the sequences from the schedule, and feeds the *Animation Handlers* with the next animation segments. After it receives a 'sequence complete' message, the next sequence is picked up from the schedule, and again fed to the handlers etc. If the schedule is empty, the *Animation Scheduler* is responsible to signal that an animation stream has been completed and to destroy the animator objects (in order to release all reserved resources). When all animation segments are completed, the *Animation Scheduler* signals the end of the conversational event and starts processing IDLE behavior. During this time, also models of subconscious behavior generation may be adapted (e.g. eye-blinks during speech vs. no speech, etc.). As a result, the *Event Handler* (if no more co-verbal events arrive) triggers the manifestation of the idle behavior.

In Panda 3D *realizer*, the frame by-frame operations are handled internally by the renderer. Thus, the *Animation Scheduler* feeds renderer with the targeted transformation (e.g. end pose), with the interpolation, and with the time interval for animation sequence as a whole. Frame-by-frame calculations are implemented automatically and internally by the renderer. This means that each animation can be controlled only on the 'sequence level'. Once the sequence had been fed to the renderer, an abrupt stop is the only possibility to immediately change/stop the animation. For a smooth continuation, therefore, the sequence fed to the *realizer* has to be played out completely. On the other hand as outlined in Fig. 3, in EVA U-Realizer realizer we are able to handle frame-by-frame operations outside the renderer. The *Animation Handler* actually handles frame-by-frame operations. It calculates the transformation for each frame separately, and then feeds the frame-to-frame transformations to the renderer. After all planed frames are carried out and the movement controllers reach their designated configuration, the *Animation Handler* triggers 'animation sequence done' event, and are destroyed afterwards. Thus, releasing all the reserved resources. This means that any animation can be modified at any stage, even during the execution of some step/sequence. Thus, for a smooth transition, the scheduler doesn't have to wait and adjust its temporal scheduling. It just has to adjust its frame-by-frame schedule, and replace it with new configurations. It can actually instantiate changes instantly as they occur. It can also insert new behavior between configurations, etc. As a result, the virtual character becomes more responsive, and can react to changes of the conversational, environmental and other contexts instantly. The agent also 'remembers' what it was gesturing prior to the excited state. Additionally, it can continue with the realization of that behavior after excited state dissipates.

4.1 Frame-by-Frame Operations and Interpolation

As already mentioned, one of the key advantages of the EVA U-Realizer is its capacity to dynamically manage realization on the frame level. The structure of an animation object as a concept is outlined in Fig. 4. Let us assume that the co-verbal act is represented by a series of animation sequences that are arranged over some time interval, and where each sequence is represented by a series of animation steps. Finally, each animation step is represented as a set of movement controllers traversing towards a targeted configuration in parallel. The frame-by-frame operations then handle, how each of these controllers is 'moved' from configuration A (Pose 1) to configuration B (Pose 2).

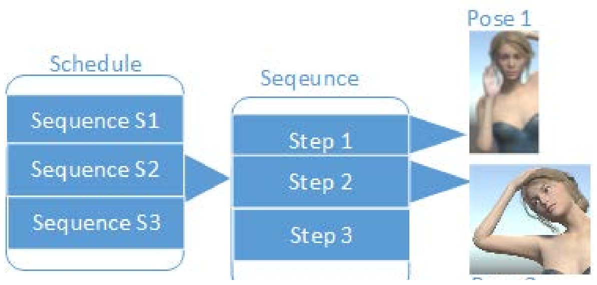


Figure 4: The representation of animation object.

In the event model, the Animation *Scheduler* picks up the first available sequence in the schedule (e.g. S1), and feeds it to the *Animation Handler*. The following pseudo-code fragment then outlines, how the *Animation Handler* actually processes and realizes the animation.

```
List<List<Transition>> AnimationSequence

while AnimationSequence haschildren
  if interrupt: break
  if previous-step-done:
    List<Transition> AniamtionStep = AnimationSequence.TakoutFirst
    foreach Transition in AniamtionStep
       AnimateTarget(Transition)
```

As can be seen, each animation sequence is represented as an object *List<List<Transition>>*, while the animation step as *List<Transition>*. Object *Transition* represents a configuration of a movement controller, by including its target and step duration. As outlined in the pseudo code, the *Animation Handler* will stop his tasks only then, if an interrupt condition is received, or if the sequence que has been cleared. For each animation step then *Animation Handler* separately processes each movement controller. This is important, since in the same step several movement controllers may move at different velocities, and may be even delayed at the beginning. The *Animation Handler* continues with the next step only after the previous one has been completely finished. Each transition is moved from its current configuration to its targeted 'in-between' configuration by calculating frames in between. The next pseudo-code outlines the calculations involved in this task:

```
float duration = step.Duration;
float local_start_time = this.localStartTimes[step.Controller];
Vector3 complete = step.Target;
Vector3 begin = step.Origin[step.Controller];
float td = (Time.time - local_start_time) / duration; // frame
joint.transform.localRotation =
    Interpoaltion(Quaternion.Euler(begin),
    Quaternion.Euler(complete), td)
```

In the given pseudo-code we are trying to control a joint, e.g. elbow joint, and to rotate it from *Pose 1* to *Pose 2*. The *duration* variable stores the complete step duration, while the *local_start_time* indicates, when the controller starts to animate relatively to the beginning of the step. By using two variables plus the current time, we can perform calculations of the interpolation parameter *td*, which is always between 0 and 1. By using the *td*, the algorithm 'knows', 1) where on the interpolation curve we are (Fig. 5), and 2) the joint's configuration prior to the step and 3) the targeted end configuration. Based on the above fragments of data we can calculate the most appropriate in-between configuration at current frame. For this we use the *Quaternion.Slerp* function. *Slerp* is here a shorthand for spherical linear interpolation[9]. The *Quaternion.Slerp* then denotes that quaternions are used for units. When *Slerp* is applied to unit quaternions, the effect is a rotation with uniform angular velocity around a fixed rotation axis. Thus, the interpolation allows us for achieving smooth and continuous animation without abrupt starts or finishes, especially when the direction changes. In essence, various interpolation curves allow us to model angular velocity at various stages of the animation. Thus, we manipulate the perceived acceleration of the displayed movement in order to make it even more natural. Fig. 5 outlines a few of the curves we support via adaptation of unity's MathFX[10].

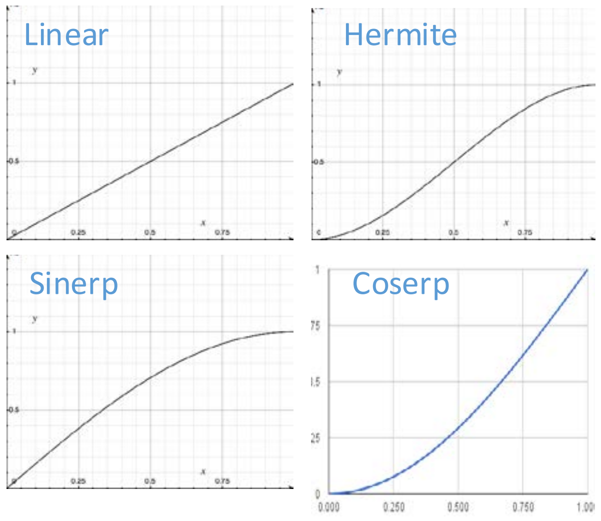


Figure 5: Linear interpolation curves.

The *linear interpolation* curve denotes that angular velocity is constant throughout the animation. Linear animation does allow for a smooth progression of the animation, however, the animation starts abruptly and ends abruptly. The *hermit interpolation* curve denotes interpolation with easing -in and -out at the limits. The *Sinerp* method interpolates, while easing around the end, when value is near one. And the *Coserp* eases in, when value is near zero, instead of easing out (and uses cosine instead of sine).

[9] https://en.wikipedia.org/wiki/Slerp

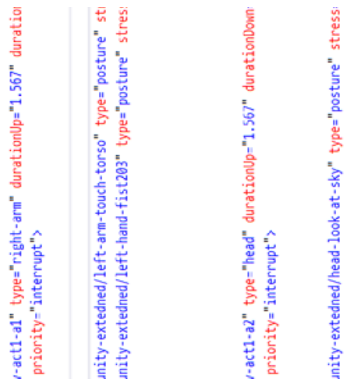
[10] http://wiki.unity3d.com/index.php?title=Mathfx

5 Results

In order to evaluate the functioning of EVA U-Realizer and to proof that all functionalities developed in Panda 3D realizer can still be supported, we have performed a series of tests. During these tests, EVA-Events were generated and realized by both realizers. The Panda 3D realizer was used first in order to validate, weather it is possible to actually realize the co-verbal act. After that EVA-Event was dispatched to the EVA-U-Realizer, where we assessed the realization in terms of smoothness, dynamics, and precision (e.g. proper controllers, to proper positions in proper time intervals). During the testing we evaluated the following concepts: a) interpretation of EVAScript language and body-part-based procedural animation, b) lip-sync, facial expressions, and animation blending. The following section outlines realization of a complex co-verbal act, performed as a part of evaluation and testing process.

5.1 Realization of Complex Co-Verbal Act

The series of tests were used to evaluate how the EVA U-Realizer interprets and realizes EvaScript events, including various layers of complexity and EVA Script attributes introduced through the definition of the EVAScript language. In this test session we (the external generator) defined a co-verbal act that consisted of two co-verbal events. Each of the co-verbal events is triggered as an event. The first one resembles the end of 'searching' idea event (e.g. when an idea of a solution comes to our mind), and the second one reassembles the beginning of revelation of the idea (e.g. how one starts outlining the solution to collocutors). The co-verbal behavior is described to be performed via full embodiment, namely, by using both arms, hands, face and head. Fig. 6 outlines the overall test scenario. However, the inner synchronization and the temporal distribution is different for each co-verbal artefact defined within the EVAScript (e.g. arms, hands, face, and head). During the first co-verbal event (e.g. revelation), the head (and face) and right hand are the dominant artefacts. Thus, they will appear to 'move' with most significance and power. On the other hand, the left hand moves to its targeted position slightly delayed, but as fast as possible. In the second act, however, the left hand is the dominant artefact. Thus, its movement will also appear as most significant, e.g. the longest and with most power. The face/head and right arm/hand are moved to the position as 'quietly' as possible. Off course, the described event is generated externally by proprietary EVA co-verbal behavior generator [15]. Thus, all expressive parameters (e.g. speed of execution, spatial configuration, power, etc.) and temporal synchronization (along with other corresponding EVA Script parameters), was pre-determined by the external generator. The EVA-U-Realizer 'simply' has to properly interpret, schedule, and realize the described animation on the specified articulated body. Fig. 6 outlines the EVAScript created for the event for the first co-verbal act, which is delivered e.g. as EVA EVENT A. Fig. 6 shows that for the first co-verbal act the overall duration of the act is 1.567s. During this time period, the agent has to perform a pointing gesture, by pointing to the sky and by moving its left arm to a position that is relevant for the specified pointing gesture (e.g. almost touching the torso). Additionally, the agent should express a blend of happy/surprised emotion on his/hers face.



```
/-act1-a1" type="right-arm" durationUp="1.567" duratio
  priority="interrupt">

unity-extedned/left-arm-touch-torso" type="posture" st
unity-extedned/left-hand-fist203" type="posture" stres

/-act1-a2" type="head" durationUp="1.567" durationDown
  priority="interrupt">

unity-extedned/head-look-at-sky" type="posture" stres
```

Figure 6: EVAScript for EVENT A, as interpreted and realized by the EVA-U-Realizer.

Fig. 7 then outlines the implementation of this complex act by using the EVA-U-Realizer. For the revelation concept, the co-verbal generator predicted that the occurrence of the idea (e.g. revelation) should be first signaled via the head movement and facial expressions, which are then followed by torso reconfiguration and pointing gesture performed in a quite fast manor. The left hand should also start to move into position at the same time; however, its movement should be significantly slower, in order to appear less relevant. As outlined in Fig. 7 head/gaze and facial expression started to appear first (delay = 0.0s). The two co-verbal artefacts then moved to their final configuration in 0.5s. The right and left-hand movement are delayed for 0.4s. Thus, both configurations started to form 0.1s interval, before the previous two co-verbal artefacts finished. The right arm finished with its animation after 0.567s, while the right hand manifested the targeted hand-shape in 0.3s. The left hand and arm propagated to their end-configuration until the overall end of the event (e.g. for 1.167s). Those co-verbal artefacts, which have already finished, just maintained their configuration.

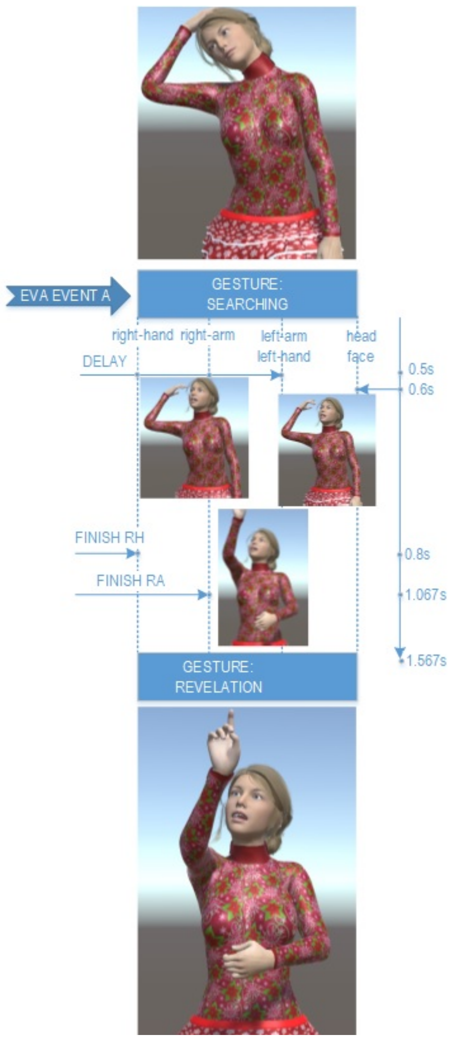


Figure 7: Realization of the co-verbal act - EVA Event A.

The second co-verbal act is delivered to EVA-U-Realizer via EVA EVENT B. Figure 8 outlines the result. During the realization of the second co-verbal act, the right arm (with hand) is regarded as less significant, therefore, it is moved to its intended rest position as slowly as possible. The left arm (with hand) is, in this situation, regarded as one of the significant co-verbal artefacts carrying some conversational meaning. The same holds true for head and face. The left hand movement in this case appeared as with most power in order to gain the most attention of the collocutor, and the face expressed confidence colored with excitement. Finally, by directing gaze to the collocutor the ECA EVA prepares the conversational environment, which facilitates the full attention of the collocutor. Thus, it can start with the representation of the recently developed idea.

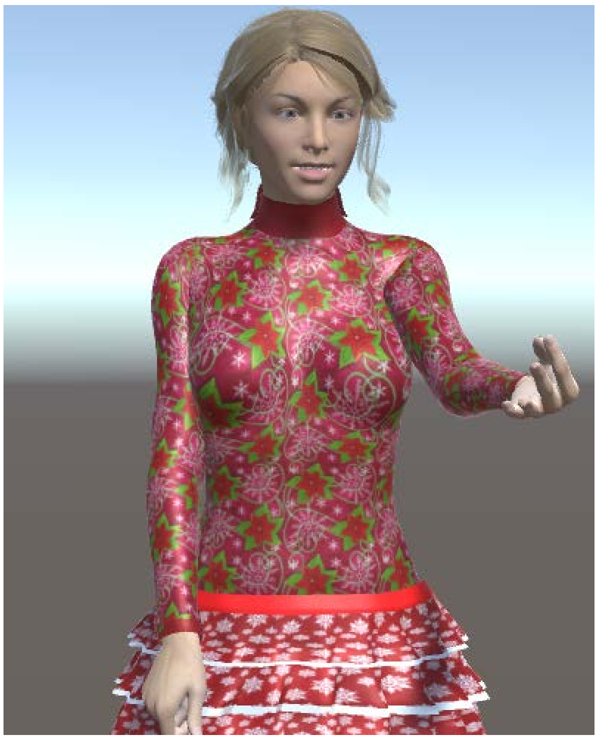
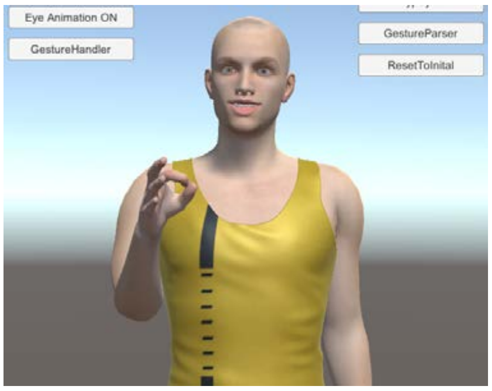


Figure 8: The realization of the second co-verbal act conversational conditions fully established, ECA establishes full attention and speaker role.

5 Discussion and Future Work

We have presented novel EVA-U-Realizer implemented in Unity 3D game engine. The realizer represents further possibilities for realizing human-like behavior on ECAs, intended especially for mobile and ubiquitous environments. The solution described allows us fully facilitate all the capacities of EVA behavior generator, and to re-use existing conversational agents and existing behavior templates. The EVA framework is able to generate affective, situation/user aware conversational behavior by synchronizing verbal and non-verbal behavior in form of conversational events. Each co-verbal event is now realizer agnostic, and can be used with various behavior realizers. With this solution, we have actually gained another arguably far more capable conversational behavior realization engine. The novel realizer is also highly modular and event oriented. Unity environment proves to be an ideal engine for rapid development of articulated agents. Namely, it doesn't require animators, or researchers specifically trained in animation skills. Namely, virtual characters Eva and Adam outlined in Fig. 3, were generated in Daz3D[11] using the default Genesis3 male and female setups. They were then imported into Unity environment via Autodesk's FBX format. The generated conversational acts (gestures) are based on the templates generated for Panda 3D based virtual character. In addition to *Asset Store* featured by Unity, researchers can rapidly integrate even other assets from various 3D modelling environments (e.g. from Maya to Blender).



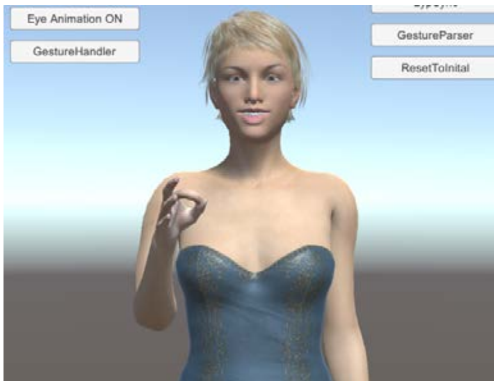


Figure 9: Adam and Eva; EVA-U-Realizer performs the same generated behavior on different embodied conversational agents

As indicated in Fig. 9, the EVA-U-Realizer also allows us to quickly develop several prototypes of an idea and test them on various ECAs. Namely, Unity also features some components usually found in modern 3D modeling framework, such as: drag & drop editing, shades, animation, and other online configuration of controllers. All these and similar mechanisms are already in place, and allow a developer to directly diving right into developing a functionality or, to test a visualization of a new

[11] https://www.daz3d.com/

conversational concept, without extra knowledge regarding 3D modeling framework. The editor's GUI, featured in Unity is also very powerful and intuitive. It allows for 'pausing' the execution, and manipulating the scene at any time, as well as progress gameplay frame-by-frame. It also has powerful asset management, and attributes inspection.

Finally, the event oriented concept designed as the part of the realizer allows the virtual agents to properly react to any given conversational context, instantly as it appears. This will enable us to integrate proprietary conversational agents into complex dynamic situations, were agents need not only to be expressive, but also to be pro-active. Such agents are not only presenters or listeners, but can also provide reactive feedback and even take over the conversation, just as real humans do. The event-oriented concept gives ECAs, the capacity to react to events instantly, and to manipulate animations at frame level.

However, Unity 3D is currently still proprietary, therefore, closed source game engine. Although, non-commercial variations are available, updates to the core may be provided only by the game engines developers. In case that performance doesn't satisfy the growing requirements of a project, or when an extra functionality is needed, it has to be implanted as an add-on to core functionality. Such solutions generally result in degraded performance, or at least in suboptimal results. Further, since Unity 3D is quite self-centered, the migration to another engine will probably require some extra work. Namely, Unity 3D uses unique approach for implementing and executing various animation/graphical concepts. Some of them are less compatible with other game engines. Also due to these facts, we support and develop further also Panda 3D realizer within the EVA framework. The realizer is currently primarily used for the desktop/server based solution.

To sum up, the ability to express, plays a central role in defining ECA's personality, its emotional state, and can produce agents as active participant in conversation. However, in order to make agents to be perceived even more natural, the agents must be able to respond to situational triggers smoothly and almost instantly, as they are perceived. Thus, the presented EVA framework seems not only more exciting, but also an important step towards generating more natural and human-like companions, and machine generated responses. Through a series of evaluation tests we have tested and shown the capacity of the novel EVA U-Realizer and its capability to properly handle co-verbal behavior generated by behavior generators in EVA Framework. Some challenges, such as generation of EVAScript behavior templates, and variety and plausibility of idle behavior, however, still remain. Also the use of lightning, shaders, and exploitation of the scene is quite rudimentary. Thus, we will work on the idea to exploit various physical objects in the scene for even more natural conversation. Through the use of Unity Animation editor, its Inverse Kinematics capabilities, and a powerful PhsyX engine (with IK), these options are viable improvements in the near future.

Acknowledgments:
This work is partially funded by the European Regional Development Fund and the Ministry of Education, Science and Sport of Slovenia; project SAIAL.

This work is partially funded by the European Regional Development Fund and Republic of Slovenia; project IQHOME.

References:

[1] Luger, E., & Sellen, A. (2016, May). Like having a really bad PA: the gulf between user expectation and experience of conversational agents. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (pp. 5286-5297). ACM.
[2] Ochs, M., Pelachaud, C., & Mckeown, G. (2017). A User Perception--Based Approach to Create Smiling Embodied Conversational Agents. ACM Transactions on Interactive Intelligent Systems (TiiS), 7(1), 4.
[3] Fabian, R., & Alexandru-Nicolae, M. (2009). Natural language processing implementation on Romanian ChatBot. In WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering (No. 5). WSEAS.
[4] Malcangi, M. (2009). Soft-computing methods for text-to-speech driven avatars. In Proceedings of the 11th WSEAS international conference on Mathematical methods and computational techniques in electrical engineering (pp. 288-292). World Scientific and Engineering Academy and Society (WSEAS).
[5] Kuhnke, F., & Ostermann, J. (2017, July). Visual speech synthesis from 3D mesh sequences driven by combined speech features. In Multimedia and Expo (ICME), 2017 IEEE International Conference on (pp. 1075-1080). IEEE.

[6] Caridakis, G., & Karpouzis, K. (2004). Design and implementation of a greek sign language synthesis system. WSEAS Transactions on Systems, 3(10), 3108-3113.

[7] Rojc, M., Presker, M., Kačič, Z., & Mlakar, I. (2014). TTS-driven Expressive Embodied Conversation Agent EVA for UMB-SmartTV. International journal of computers and communications, 8, pp. 57-66.

[8] Tolins, J., Liu, K., Neff, M., Walker, M. A., & Tree, J. E. F. (2016). A Verbal and Gestural Corpus of Story Retellings to an Expressive Embodied Virtual Character. In *LREC*.

[9] Esposito, A., Esposito, A. M., & Vogel, C. (2015). Needs and challenges in human computer interaction for processing social emotional information. Pattern Recognition Letters, 66, 41-51.

[10] Kok, K. I., & Cienki, A. (2016). Cognitive Grammar and gesture: Points of convergence, advances and challenges. Cognitive Linguistics, 27(1), 67-100.

[11] Kopp, S., & Bergmann, K. (2017, April). Using cognitive models to understand multimodal processes: The case for speech and gesture production. In The Handbook of Multimodal-Multisensor Interfaces (pp. 239-276). Association for Computing Machinery and Morgan & Claypool.

[12] Pelachaud, C. (2015, May). Greta: an interactive expressive embodied conversational agent. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (pp. 5-5). International Foundation for Autonomous Agents and Multiagent Systems.

[13] Neff, M. (2016). Hand Gesture Synthesis for Conversational Characters. Handbook of Human Motion, 1-12.

[14] Rojc, M., Mlakar, I., 2016. An Expressive Conversational-behavior Generation Model for Advanced Interaction within Multimodal User Interfaces, (Computer Science, Technology and Applications). Nova Science Publishers, Inc., Corp., New York, 234.

[15] Rojc, M., Mlakar, I., & Kačič, Z. (2017). The TTS-driven affective embodied conversational agent EVA, based on a novel conversational-behavior generation algorithm. Engineering Applications of Artificial Intelligence, 57, 80-104.

[16] Gratch, J., Hartholt, A., Dehghani, M., & Marsella, S. (2013). Virtual humans: a new toolkit for cognitive science research. Applied Artificial Intelligence, 19, 215-233.

[17] Thiebaux, M., Marsella, S., Marshall, A. N., & Kallmann, M. (2008, May). Smartbody: Behavior realization for embodied conversational agents. In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1 (pp. 151-158). International Foundation for Autonomous Agents and Multiagent Systems.

[18] Pelachaud, C., 2015. Greta: an interactive expressive embodied conversational agent. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, (pp. 5-5).

[19] Klaassen, R., Hendrix, J., Reidsma, D., & van Dijk, B. (2013). Elckerlyc Goes Mobile Enabling Natural Interaction in Mobile User Interfaces.

[20] Heloir, A., & Kipp, M. (2010). Real-time animation of interactive agents: Specification and realization. Applied Artificial Intelligence, 24(6), 510-529.

[21] Kolkmeier, J., Bruijnes, M., Reidsma, D., & Heylen, D. (2017, August). An asap realizer-unity3d bridge for virtual and mixed reality applications. In International Conference on Intelligent Virtual Agents (pp. 227-230). Springer, Cham.

[22] Mlakar, I., & Rojc, M. (2011). EVA: expressive multipart virtual agent performing gestures and emotions. International journal of mathematics and computers in simulation.

[23] Bédi, Branislav, et al. "Starting a Conversation with Strangers in Virtual Reykjavik: Explicit Announcement of Presence." Proceedings from the 3rd European Symposium on Multimodal Communication, Dublin, September 17-18, 2015. No. 105. Linköping University Electronic Press, 2016.

[24] Li, J., Galley, M., Brockett, C., Spithourakis, G. P., Gao, J., & Dolan, B. (2016). A persona-based neural conversation model. arXiv preprint arXiv:1603.06155.

[25] Gibet, S., Carreno-Medrano, P., & Marteau, P. F. (2016). Challenges for the Animation of Expressive Virtual Characters: The Standpoint of Sign Language and Theatrical Gestures. In Dance Notations and Robot Motion (pp. 169-186). Springer International Publishing.

[26] Salama, M. A. R. I. A., & Shawish, A. H. M. E. D. (2013). A Comprehensive Mobile-Based Companion for Diabetes Management. In 7th WSEAS European Computing Conference, Dubrovnik.

[27] Ochs, M., Pelachaud, C., & Mckeown, G. (2017). A User Perception--Based Approach to Create Smiling Embodied Conversational Agents. ACM Transactions on Interactive Intelligent Systems (TiiS), 7(1), 4.

[28] Belessiotis, V. S., Vosinakis, S., & Panayiotopoulos, T. (2001). The use of the Virtual Agent SimHuman in the ISM scenario system. Advances in Automation, Multimedia and Video Systems, and Modern Computer Science, 97-101.

[29] Linehan, C., & McCarthy, J. (2017). 57 Using conversation to model interaction in the MATHS workstation. Engineering Psychology and Cognitive Ergonomics: Volume Five-Aerospace and Transportation Systems, 51.

[30] El Haddad, K., Cakmak, H., Dupont, S., & Dutoit, T. (2016). Laughter and Smile Processing for Human-Computer Interactions. Just talking-casual talk among humans and machines, Portoroz, Slovenia, 23-28.

[31] Linehan, C., & McCarthy, J. (2017). 57 Using conversation to model interaction in the MATHS workstation. Engineering Psychology and Cognitive Ergonomics: Volume Five-Aerospace and Transportation Systems, 51.

[32] Porcheron, M., Fischer, J. E., McGregor, M., Brown, B., Luger, E., Candello, H., & O'Hara, K. (2017, February). Talking with conversational agents in collaborative action. In Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (pp. 431-436). ACM.

[33] Pérez, J., Sánchez, Y., Serón, F. J., & Cerezo, E. (2017, August). Interacting with a semantic affective ECA. In International Conference on Intelligent Virtual Agents (pp. 374-384). Springer, Cham.

[34] Kang, J., Badi, B., Zhao, Y., & Wright, D. K. (2006, April). Human motion modeling and simulation. In 6th International Conference on Robotics, Control and Manufacturing Technology (ROCOM 2006) (pp. 62-67).

[35] Akinjala, T. B., Agada, R., & Yan, J. (2016, December). Animating Human Movement & Gestures on an Agent Using Microsoft Kinect. In Multimedia (ISM), 2016 IEEE International Symposium on (pp. 369-374). IEEE.