The Design of a Generic Framework for Integrating ECA Components

Hung-Hsuan Huang, Toyoaki Nishida
Graduate School of Informatics, Kyoto University, Japan
huang@ii.ist.i.kyoto-u.ac.jp, nishida@i.kyoto-u.ac.jp

Aleksandra Cerekovic, Igor S. Pandzic
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
{aleksandra.cerekovic, Igor.Pandzic}@fer.hr

Yukiko Nakano
Department of Computer, Information and Communication Sciences, Tokyo University of Agriculture & Technology, Japan
nakano@cc.tuat.ac.jp

ABSTRACT

Embodied Conversational Agents (ECAs) are life-like computer generated characters that interact with human users in face-to-face multi-modal conversations. ECA systems are generally complex and difficult for individual research groups to develop. Therefore, if there was a common framework for connecting ECA functioning blocks seamlessly to an integrated system, redundant effort can be saved. This paper discusses the issues emerged in developing such a framework and introduces the design and preliminary results of our ongoing project, Generic ECA Framework.

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Human factors; I.2.0 [**Artificial Intelligence**]: General

General Terms

Design, Human Factors

Keywords

Embodied Conversational Agent, Blackboard, Application Framework

1. INTRODUCTION

Embodied Conversational Agents (ECAs) are life-like computer generated characters that interact with human users in face-to-face multi-modal conversations. Without the limitations due to the clumsy physical bodies built by today's technologies, they can provide richer facial expressions and larger degree of freedom in their bodies with higher precision than contemporary humanoid robots. They thus allow the researchers to concentrate on building advanced virtual humans with the capability to perform complex high-level communicational functions, such as turn taking via subtle eye contacts or realistic facial expressions for expressing emotional states.

Cite as: The Design of a Generic Framework for Integrating ECA Components, H.-H. Huang, A. Cerekovic, I. S. Pandzic, Y. Nakano and T. Nishida, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons(eds.),May,12-16.,2008,Estoril,Portugal,pp. 128-135.

ECAs are ideal for simulations in psychology experiments, training or education applications, and entertainment purposes. They also provide more nature and richer interface rather than widely used speech-only ones of the systems designed for public users who are not to familiar with the operation of computers. ECAs have been proven to be powerful tools for engaging human users in computer-interaction, augment their involvement and reduce their workload [2]. Because the variety of potential applications, ECAs attract great interests from various fields.

However, to realize a believable ECA capable to take out natural multi-modal face-to-face conversations with humans is not an easy task. In addition to the prosody properties of verbal channel, precise control on non-verbal channels like gazing, raising of eyebrows, nod, hand gestures or postures are required. These features are used to perform communicative functions like directing the flow of conversations or supplements of verbal utterances while appropriately reflecting the agent's internal emotional states, personality and social status in responding to the recognized attentions from human users by sensory devices. And then, those behaviors are output with realistically rendered characters, virtual environment as well as fluent speech synthesis. To achieve these functionalities, knowledge and techniques on signal processing, natural language processing, gesture recognition, artificial intelligence, dialog management, personality and emotion modeling, natural language generation, gesture generation, CG character animation and so on are required.

ECA development involves so many research disciplines that it is difficult for individual research teams to develop from the scratch. The usual way to build ECA systems is thus by utilizing software tools developed by other institutes. However, software tools developed by different institutes are usually neither meant to cooperate with each other nor designed for the same application domain, it is usually laborious or even impossible to make them work with each other. Moreover, due to the common purposes, redundant effort and similar approaches are repeated by the researchers.

To relief these problems, if there was a common framework that absorbs the heterogeneities and connects diverse ECA software tools and drives the connected components as an integral ECA system, redundant effort and resource uses can be saved, the sharing of research results can be facilitated and the development of ECA systems can become easier.

This paper discusses the issues emerged in developing such framework and introduces the design and preliminary results of our ongoing project, Generic ECA Framework.

2. ECA COMPONENT INTEGRATION ISSUES

Like typical modeling of autonomous agent systems, ECAs need to posses the following capabilities:

- Acquire verbal and nonverbal inputs from the human user and the environment
- Interpret the meaning of the inputs and deliberate the responding verbal and nonverbal behaviors
- Render those behaviors with computer graphics character animations as outputs

To realize these capabilities, various functionalities like sensor data acquiring, speech recognition, gesture recognition, natural language understanding, believe-desire-intention planning, speech synthesis, CG animation renderer and so on are required. ECA developers then need to implement these functionalities by their own or by utilizing available software tools and then integrate them to work as an ECA in certain architectures. Here, we call the modules that handle each individual functionality as the *Components* of the system. ECA component integration involves various issues, and some of them are already mentioned in [4]. To achieve a common ECA component integration framework for general purposes, several requirements need to be fulfilled:

Modularity and reusability. This should be the heart of any integration approach. Component reusability can be maximized by cleanly divided functionalities of components and clearly defined interface between each other. Simpler functionalities handled by each component and lower interdependency improve modularity.

Distributed architecture and OS/programming language independence. Components may be developed by various programming languages and run on various operating systems. The ability of the integration framework to provide supports for major operating systems and programing languages allows the connected components to run on multiple machines is a necessity.

Support of various natural languages. With the advance of transportation, the world becomes smaller and smaller and cross-culture issues have been emerging much more importance than before. However, due to the fact that western countries dominate the development in the field of computer science, the support and issues related to Asian languages or others are often ignored. To achieve generality of the common framework, the flexibility to handle various languages and cultures need to be maintained.

Two-way communication among components. The ECA components do not only "pull" data from the others, but some of them such as sensor data processing components a"push" data to the others. A mechanism which supports such two-way data passing is required.

Fusion of multi-modal inputs. In multi-modal interactive ECA systems, the relationship between user inputs coming from speech channel or other sensor channels needs to be identified correctly and is used to trigger appropriate agent responses.

Virtual environment control. Not only the virtual character themselves but also the virtual environment where they are in need to be altered corresponding to the interactions between the agent and the human user, e.g. scene change or camerawork.

Real-time performance and timing control. Real-time response of the agent to user's inputs is one of the basic requirements of ECA systems. The latency of each part of the system needs to be kept as minimum while on-time execution of actions need to be guaranteed. Therefore, a strict temporal model is necessary.

Synchronization between prerecorded tracks and run-time generated behaviors in outputs. Fixed length prerecorded tracks including voice, music, or motion captured animation sequences need to be synchronized with each other and run-time generated actions.

Synchronization between verbal and nonverbal behaviors in outputs. Verbal and nonverbal behaviors are interrelated, supple each other and need to be synchronized.

Ease the efforts to adopt legacy systems. Libraries or tools need to be provided to ease the effort to develop wrappers for legacy systems to be connected to the framework.

User interruption. Provide the flexibility that allows smarter system to modify its current behaviors on-line instead of simply stopping them and then launch the new ones.

Capability to support subtle spontaneous reactive behaviors. The support of spontaneous and subtle reactive feedback behaviors like gaze, nods or back-channel utterances can improve the believability of the agent.

Consistency. All of the output modalities need to be consistent with the agent's internal state, for example, if the agent is in happy mood, it is not supposed to speak in a angry tone or has a sad facial expression.

3. THE DESIGN OF GENERIC ECA FRAMEWORK

The Generic Embodied Conversational Agent (GECA) Framework has three parts, the integration backbone GECA Platform, communication libraries GECA Plugs, and a high level protocol GECA Protocol. Figure 1 shows the diagram of the GECA Framework's basic concepts.

3.1 The Basic Architecture of the GECA Framework

ECA is not a new research area, and there are many excellent individual ECA systems, and various integration architectures have been proposed. However, contemporary ECA architectures are usually designed for specific applications, and their architectures like REA [3] typically feature fixed processing pipelines of functional components and thus can not be easily adapted to other applications. On the other hand, blackboard model is a methodology widely used in distributed and large-scale expert systems. Its basic idea is the use of a public shared memory where all knowledge sources read and write information. The interdependency among the knowledge sources can be minimized, and thus it is suitable for integrating heterogeneous knowledge sources. Considering black board's convenience and generality in integrating various components, we adopted it as the basic architecture of GECA Platform.

In GECA, multiple blackboards are allowed. Components

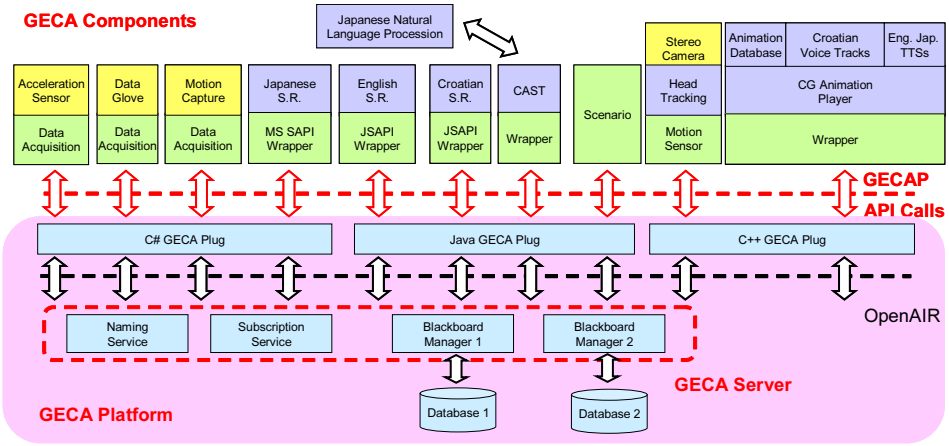

Figure 1: The conceptual diagram of GECA Framework and the configuration of a multimodal tour guide agent. The pink square indicates the communication backbone

connecting to those blackboards share data in subscribe-publish message passing mechanism. Every message has a message type, when a specific message is published to a blackboard, it is forwarded to the components which subscribed the corresponding message type. Those components then process the received message and publishes the results as their contribution to the blackboard. To reduce the overhead of message forwarding, direct communication between components is allowed. Every blackboard has its own manager, and there is a server that provides message subscription and naming services for the whole system.

There are many available technologies for implementing distributed systems, but most of them suffers from certain drawbacks and are not appropriate in the ECA context. For example, KQML does not provide explicit temporal control, CORBA, Web Service and remote procedure invocation do not support two-way data passing. Therefore, a simple and light weight protocol, OpenAIR[1] was chosen as the low-level routing protocol for the communication among components, the server and blackboards.

OpenAIR is a specification of XML message passing for distributed systems in a TCP/IP network. We considered that it is suitable for real-time interactive system because its very simple message format and specific features for real-time systems such as the explicit timestamps. A Java reference implementation of its library called Plug is freely published. The second part in the GECA Framework is thus called GECA Plug libraries. They are extended OpenAIR Plug with GECA's original classes and functions. Currently, C#, C++ versions have been developed while the Java version is modified from the reference implementation. The purpose of the GECA Plugs is to absorb the differences caused by operation systems and programming languages, and to make system development easier. By utilizing GECA Plugs, an ECA developers only need to implement a small wrapper for an existing software tool then it can be plugged into the framework and cooperates with the others.

The third part of the GECA Framework is the GECA Protocol, it is a specification of available message types and high-level XML message formats that are transferred on the GECA Platform. The detailed introduction of this protocol is left to section 3.2.

Comparing to previous architectures, GECA Framework is expected to have the following advantages:

- Components developed with different programming languages and running on different OS's can be integrated easily
- Components which require heavy computation can be distributed to multiple computers to improve the overall system performance
- The single-layer component hierarchy shortens decision making path and eases the support of reactive behaviors
- Explicit temporal information and synchronization specifiers ensures that components are synchronized
- ECA systems with various features can be configured easily with different component topologies
- The weak inter-dependency among the components allows on-line switching and upgrading of components
- Direct component communication multiple blackboards can lower message transmission load

3.2 GECA Protocol

Based on the low-level communication platform of GECA Framework, GECA Protocol (GECAP) is an XML based high-level communication protocol among the components. In GECAP, every message has a type, for example, "`input.action.speech`" represents a speech recognition result, "`output.action.speech`" represents a text string to be synthesized by a Text-To-Speech (TTS) engine, etc. Each message type has a specified set of elements and attributes, for example, "`Intensity`", "`Duration`", "`Delay`", etc. All data is represented as plain text and transferred in OpenAIR on the GECA platform. GECAP is a specification of message format style and a set of core message types, the syntax is not fixed and can be easily extended to meet the demands of individual applications.

[1]http://www.mindmakers.org/openair/airpage.jsp

CGECAP message types can be divided into three categories: input phase, output phase, and system messages. Input and output messages can be further categorized into three layers, raw parameter, primitive action, and semantic interpretation in the sense of abstractness.

GECAP Message Types in Input Phase. The task of the components which generate input message types is to acquire and to interpret human users' inputs from multi-modal channels. The followings are some examples of defined input message types where "`input.action.*`" types transfer primitive actions and "`input.raw.*`" types transfer raw parameters. "`input.action.speech`" for speech recognition result, "`input.action.head`" for head movements such as nodding and shaking that can be detected by an acceleration sensor, "`input.action.gaze`" for gaze direction that can be approximated by a head tracker, "`input.raw.hand`" for hand shapes acquired by data glove devices, "`input.raw.arm`" for the angles of the arm joints that can be approximated by three motion capture sensors attached on each arm, "`input.action.gesture`" for predefined hand gestures that is recognized by motion capturing devices, "`input.action.point`" for pointing gestures which can be detected by a motion capturer or even a mouse.

The following is an example of an "`input.action.speech`" type message. This message type also utilizes the language attribute of content slot of OpenAIR to store the recognized natural language with values like "English."

```
<Perception Begin="1175083369171"
    Duration="500" Weight="1.0">
    <Hypothesis Confidence="0.9" >
        <Speech>what is this</Speech>
    </Hypothesis>
    <Hypothesis Confidence="0.1" >
        <Speech>what is these</Speech>
    </Hypothesis>
</Perception>
```

The recognized result is stored in the `Speech` element. Programs like speech recognizer or gesture recognizer usually have ambiguity in recognizing the data from sensors. The `Hypothesis` element presents a list of hypotheses of the recognition result wintin a single input event with confidence ratings in values from 0 to 1. `Begin` attribute stores when this input event begins with the absolute time represented in milliseconds while `Duration` attribute stores how long the input event lasted. The following is an example of an "`input.action.point`" type message that represents a position on the 2D screen where the user is pointing by performing a pointing gesture or by using a pointing device:

```
<Perception Begin="1175079954578"
 Duration="2000" Weight="0.5">
    <Hypothesis Confidence="1.0">
        <Point X="0.2" Y="0.3"/>
    </Hypothesis>
</Perception>
```

GECAP Message Types in Output Phase. The only actuator of software based ECAs is the character animation player. This player plays text strings as voice with TTS and drives the CG character to move in the virtual environment when a command message arrives in real-time. Although current prototype of GECA player is implemented with a commercial software, Visage|SDK[2], the design of GECA's output message format is not bound to Visage and should be able to be ported to other animation systems. All parts of the full 3D anthropomorphic character like the limbs, fingers, eyes, mouth can be animated to perform arbitrary actions that are possible for a real human. The animation player also provides the support of MS SAPI compatible TTS engines for the character's speech. To simplify the problem and also because a picture looks more realistic than a full 3D environment which lacks enough details, the virtual environment for the agent's activities is currently represented by switching 2D background images.

The following multi-modal utterance is an example of the content of the "`output.action.multimodal`" message that is accepted by the animation player. A well implemented TTS engine adjusts its intonation output in the unit of sentences rather than just speak out words. In order to take advantage of this feature, the agent's utterances are broken into sentences according to the punctuation marks. Sentences are then enclosed with `Sentence` and `Utterance` elements before they are sent to the player or the other components. Sentences are expected to be the basic unit that will be executed by the animation player. The ECA's non-verbal behaviours are described in the `Action` elements, and their timing information is encoded by the containing relationship with the verbal `Speech` elements. TTS engines' prosody information specifying tags are not a part of GECAP but they are allowed to be inserted into the `Utterance` element. GECA component will ignore them and pass them to be processed by the TTS. The detailed introduction of the `Action` element is left to section 3.3.

```
<Utterance>
   <Sentence><Speech>Hello.</Speech>
   </Sentence>
   <Sentence><Action Type="expression"
    SubType="smile" Duration="2300"
    Intensity="0">
   <Action Type="bow" Duration="1700"
    Intensity="1"
    Trajectory="sinusoidal"/>
   <Speech>My name is Dubravka and I
    will</Speech>
   <Action Type="beat" SubType="d"
    Duration="600"/>
   <Speech>be your tour guide agent of
    Dubrovnik city.</Speech></Action>
   </Sentence>
</Utterance>
```

Since the character's body model and animations are presented in MPEG-4 standard, raw data parameters for the reference Visage player are MPEG-4 Face and Body animation parameters (FBAs). Message type "`output.raw.FBAP`" is defined to carry the used parameters' numeric value and drive the character in real-time. Figure 2 shows an example system where the user avatar and the computer controlled agents are driven in real-time by "`input.raw.arm`" and "`output.raw.FBAP`" messages.

[2]http://www.visagetechnologies.com

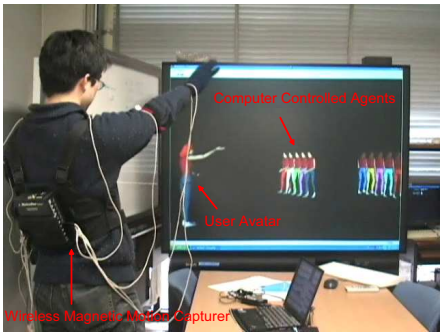

Figure 2: A culture difference experiencing application with 1 user avatar and 10 computer controlled agents driven by raw parameters to raw parameters

System Message Types. There are system controlling message types such as "`system.status.player`" or "`system.control.player`" to query the status of the ECA character (e.g. whether the character is speaking something) or make the character to stop speaking and playing any animation, etc.

3.3 GECA Scenario Mark-up Language

Considering the complexity and the fact that current technology is still impossible to drive an ECA to behave like a human in an indistinguishable level, instead of a block of complex deliberate process, we have defined a script language, GECA Scenario Mark-up Language (GSML) that defines the user-agent interactions. A script definable ECA is less general than a deliberative process, but it will be much easier to create contents and should be useful enough for simpler ECA interface applications.

GSML is inspired by AIML[3] that is a popular script language for defining text based chatbots on the Web. An AIML script represents an agent's knowledge that is composed by a set of `Category` elements. One `Category` contains a pair of `Pattern` and `Template` that describes one of the possible conversations between a chatbot and a human user. When a user's utterance comes into the interpreter, it is matched with all defined patterns to find a corresponding `Template` which describes agent's responses to the utterance. However, AIML can not be applied to the ECA context due to the following reasons: it supports English only, unexpected template may be triggered because the same patterns can not be distinguished in different circumstances, it can not describe non-verbal behaviors of neither human user nor agent, there is no way to specify objects in the virtual world, agent behaviors need to be triggered from the human side.

GSML extends AIML's syntax to cover more complex situations in ECA-human conversations. Extending to AIML's one-layer categories, GSML represents human-ECA conversations as conversational states and the transitions among them. Figure 3 shows the additional three layers of the hierarchy of GSML categories. In GSML, one `Scenario` defines an interactive scenario between the ECA and the human user. A scenario can contain one or more `Scene` elements while each `Scene` means a physical location in the virtual world and has an ID specifying its settings. In current player implementation, this ID is coupled with a background image. In an individual, there may be one or more `State` elements. Each `State` contains one or more `Category` elements. The states are linked by `Transition` specifications described in `Template` elements. Templates can be triggered right away when conversational state transition occurs even without user inputs. The Scenario-Scene-State-Category hierarchy narrows the range of possible categories into a conversational state and prevents the problem that templates may be triggered unexpectedly in AIML agent which practically has only one conversational state. Besides, the `Language` attribute in states allows a multi-lingual ECA to be defined in a single GSML script.

GSML's patterns and templates do not only present verbal utterance of the agent but are also extended to describe non-verbal behaviors of the agent and the human user. `Action` tags that specify face or body animations can be inserted into the utterances of the agent, the timing information is specified by the position of the `Action` tags in the utterance texts. The action tags (`Speech`, `Point`, etc) can be inserted inside the `Pattern` tags then the corresponding template will be triggered if the user does that non-verbal behavior. Further, particular areas of the background image can be named by `Object` elements and can be referred (e.g. pointed at or gazed at) by the user during the multi-modal conversation.

By observing usual face-to-face communications between humans, we can find non-verbal behaviors are the indispensable counterpart of verbal utterances. For example, the verbal utterance "What is this?" with a pointing gesture is a very typical example. Without the pointing gesture, which object that this "this" is mentioning becomes ambiguous. On the other hand, a pointing gesture can not fully convey the user's intention, either. Generally, the order, combination, and occurrence of multi-modal perceptions and their relationship are difficult to be described and identified. Although the conversation structure of GSML is similar to classic transition network approaches like [5], it additionally incorporated the features dedicated to multi-modal human-agent interactions. As the discussion in the specification of W3C's multi-modal interface description language for Web browsing, EMMA[4], it is not appropriate to propose a general algorithm for multi-modality fusion. In GSML and its interpreter (the scenario component), we adopted a simplified description for multi-modal perception of the ECA and a relatively simple mechanism to solve reference ambiguities. Since EMMA is designed for similar purpose as GECAP's input phase and GSML, some of the element names that we are using are inspired from some element names in EMMA, however, what do they mean and how they are used are very different to their counterparts in EMMA.

`Set` element means a non-ordered set of multiple verbal or non-verbal perceptions and every one of them must be fulfilled. `OneOf` element means at least one of the multi-modal perceptions needs to be fulfilled. `Sequence` means the multi-modal perceptions need to be performed by the human in the specified order. The three specifiers can be further nested with each other. Whether two multi-modal perceptions occur concurrently is judged by the period coverage of involved perceptions according to the `Begin` and `Duration` attributes in the message sent from the sensor data acquiring components. The scenario component keeps a current status of the multi-modal perceptions and triggers the corresponding

[3]http://www.alicebot.org

[4]http://www.w3.org/tr/emma

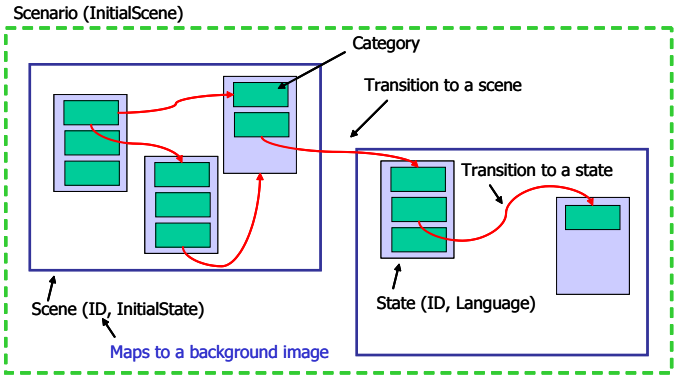

Figure 3: The diagram showing the relationship between`Scenario`, `Scene`, `State`, and`Category` elements in GSML

`Template` if any one of the available patterns defined in the current conversational state can be exactly matched. This matching is calculated every time when a new input message arrives. The combination which has highest value of the sum of the product of confidence and component weight is chosen in the matching. The following is an example code segment describing the interaction between the human user and a tour guide agent at the entrance of the Dubrovnik old town.

```
<Scene ID="Entrance" InitialState="Greet"
  X="1250" Y="937">
  <Objects><Object ID="Fountain" X="900"
    Y="0" Width="350" Height="937"/>
    <Object ID="Monastery" X="0" Y="0"
      Width="377" Height="937"/>
  </Objects>
  <State ID="Greet" Language="English">
    <Category><Pattern>
      <Speech>hello</Speech></Pattern>
      <Template>Hello, my name is
         Dubrovka, and I am the guide here.
         Where do you want to go at first?
         <Action Type="pointing"
         Duration="1000" Direction="right">
         The fountain</Action>or<Action
         Type="pointing" Duration="1000"
         Direction="left">the monastery?
         </Action></Template></Category>
    <Category>
      <Pattern>
        <OneOf>
          <Speech>fountain</Speech>
            <Set><Speech>I want to go there
              </Speech>
              <Point Object="Fountain"/>
            </Set>
        </OneOf>
      </Pattern>
      <Template>Please follow me here.
        <Transition Scene="Fountain">
      </Template>
</Category>......
```

The fore part of this code specifies the scene with an ID, "`Entrance`." The `Object` elements specify two areas of the background image, "`Fountain`" and "`Monastery`." These areas are used to in the matching of the coordinates sent from some pointing component with the `Object` specifiers in second `Category`. According to the description of perception specifiers, when either one of the two conditions is fulfilled, a conversational state transition to the initial state of the scene, "`Fountain`" will be triggered. When the human user says "fountain", or when the user says, "I want to go there" while performing a pointing gesture on the screen where the position is recognized as an `X` value from 0.72 to 1.0 and a `Y` value from 0 to 1.0 at the same time.

The `Action` elements are the specifiers of non-verbal animations of the ECA character. The timing to start to play the specified animation is determined by the position of the opening tag relative to the verbal utterance. In the case when the agent does not say anything, a `Delay` attribute is used to specify when the animation will be played relative to the beginning of the template. This attribute can also be used to play the overlapping actions which can not be directly represented by tag position and coverage relationship. The playing of this animation will end when the agent speaks to the closing tag of `Action` element or meets the time specified by the `Duration` attribute. `Subtype` specifies another action in the same category if available. `Intensity` specifies the strength of an action if it is specifiable. `X`, `Y`, and `Z` specify a position in the virtual world if the action has a destination, e.g. walking, pointing, gazing actions. `Direction` specifies a direction of the action if available. `Trajectory` specifies the temporal function to change parameter values in playing the animation, "Linear", "Sinusoidal" and "Oscillation" are currently available values. `Sync` attribute specifies the temporal relationship between the actions in an utterance. There are three possible values: "WithNext", "BeforeNext", and "PauseSpeaking" stand for do not wait for this action, to wait for this action to end, and to pause TTS while executing this action respectively. A template is transferred as an `Utterance` element in GECAP, the contents of it is broken into text chunks and sentences as described in section 3.2.

Since there is no reasonable boundary for possible actions that can be done by a human or an ECA character, we are not going to specify a full set of the actions but only defined the syntax to specify the animations and a set of

animations that are supposed to be most frequently used. The set of available animations should be application dependent. A special action type created is the `PlayTrack` action, this action plays a background music track, voice track, or a pre-defined animation track. It can be used to implement an ECA system in a language which has no available TTS engines. For example, an agent speaking Croatian can be implemented with pre-recorded human voice tracks and synchronized lip animations. The `Delay` attribute can be utilized in this case to synchronize the tracks with each other. GSML (and output phase of GECAP) provides the distinguishing features include word-level precisely aligned non-verbal behaviors to speech channel and multi-language support. However, the non-verbal actions are intentionally kept in high-level, just a name and a set of run-time configuration parameters for maximum compatibility with various animators.

4. IMPLEMENTATION AND USES

The first development of the GECA server has been implemented in Java and the backboard is implemented on regular relational databases (MySQL[5]). New components running on multiple computers are added and connected to the GECA server through C#, C++ or Java GECA Plugs. So far, we have also implemented several different ECA applications, in which we have introduced standard GECA components such as Japanese spontaneous gesture generator [7], head tracker [8], hand shape recognizer, head pose recognizer, scenario interpreter, speech recognizer and the CG animator.

An application for experiencing cross-culture gesture differences. The avatar replays the user's hand gestures such as beckoning while ten computer controlled agents react to those gestures pretending that they are Japanese or British. The user's actions are captured by a magnetic motion capturing device and interpreted to low-level joint angles to drive the avatar character in real-time. The computer controlled agents are driven by individual reactive controlling components and a common animation catalog component. They are driven by low-level MPEG-4 BAPs in real-time, too. Figure 2 is a screen shot of this application.

A Dubrovnik city tour guide agent who interacts with a human User in multiple modalities. Most part of this system is developed as a student project during the four-week period of an international workshop, eNTERFACE'06[6]. Its improved version is currently running in three language modes, English, Japanese and Croatian. Due to the absence of reliable Croatian synthesis and recognition engines, Croatian speech output is achieved with prerecorded voice files and lip animation tracks; Croatian speech recognition is done by English speech recognizer customized with a grammar rule set covering a limited range of Croatian vocabularies. The hardware configuration and data flow of this system is shown in Figure 4 and Figure 5 respectively. In the tour guide system, the user can use natural language speaking, pointing gesture, gazing and nodding/shaking of head to interact with the agent to navigate scenes in Dubrovnik city where the whole old town is an UNESCO world cultural heritage. This tour guide agent shows the framework's capability to seamlessly deal with multi-modal inputs and sufficient performance for real-time conversations.

[5]http://www.mysql.com
[6]http://enterface.tel.fer.hr

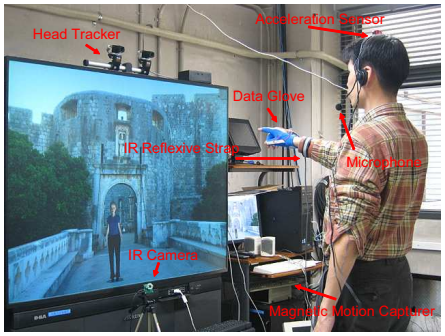

Figure 4: The multi-modal tour guide agent. Pointing position can be detected by an optical motion capturer or a magnetic motion capturer

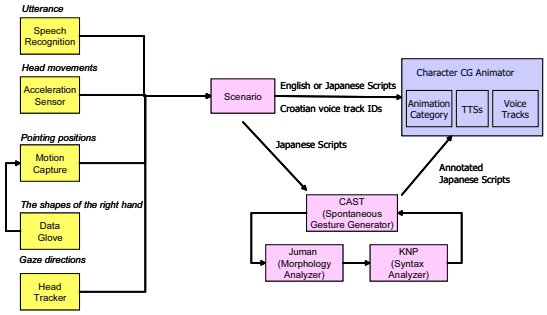

Figure 5: The data flow and component configuration of the multimodal tour guide agent. The programs, CAST, Juman and KNP communicates with each other in their original protocols

A quiz game kiosk about the knowledge of food science. This is a joint project with the National Food Research Institute (NFRI) of Japan. The expression of the agent issuing quizzes in the kiosk and the background music are controlled by an emotion component based on [1]. For example, the agent may smile when the visitor pressed a correct answer and show a bored face if there is no input for a long period. Since this system is targeted for public visitors of the showroom so that the user interface is limited to a touch panel to prevent unexpected operations. This quiz kiosk have been displayed in two open lab events of NFRI in 2007 and was the first real world use of GECA agents. In these two events, there were nearly 400 visitors played with the kiosk and initiated more than 100 game sessions in total. The kiosk was so successful in gathering visitors' attention that a constant setup of this kiosk in the showroom of NFRI is being considered.

5. CONCLUSIONS AND FUTURE WORKS

This paper represented the Generic Embodied Conversational Agent (GECA) Framework that covers the information process from the detection of the human users to the behavior outputs of the ECA. A script language (GSML) that specifies an ECA's behavior is introduced. Three example systems for preliminary evaluations are also introduced. The ultimate goal of this project is to make the framework

publicly available with a reference ECA toolkit which can be used to build ECA systems in an instant and can be extended easily.

A joint research work called SAIBA[7] has been launched to develop common framework of ECA outputs eliminate the redundant works of the researchers. The use of two languages, FML (Function Markup Language) describing agent intentions and BML (Behavior Markup Language) [6][9] describing CG character animations are being specified. Their general goal is similar to our one in output phase, but GECA additionally concerns on the implementation and integration problem and tries to cover the whole process of human-agent interaction starts from human user inputs. The initiation of BML is strong and we hope its specification can be more concrete and really become a common standard in near future. We are currently considering how to propose an integration of BML in the GECA architecture, for example, since the output phase messages in GECA use a higher level representation, it is possible to develop a compiler to transfer `Action` tags' positions of GECA to be presented by BML's synchronization points. On the other hand, the concept and specification of FML is not clear yet, and we will watch its progress.

To develop a thoroughly generic ECA framework is extremely difficult and perhaps is an impossible task. In this paper, we do not stress that we have implemented a generic ECA framework but described our approach toward this ultimate goal. If the integration framework must be generic, what kind of problems will occur and how we try to solve them, etc. Also, an overall evaluation of such a framework is difficult and impractical, for example, how can we define an objective benchmark on how generic it is or how much effort that it can save comparing to an implementation without the framework. This may be an interesting issue, but instead of that, we would like to have a brief discussion on the pros and the cons at this time point. The obvious benefit is reusability, as described in section 4, standard components such as scenario interpreter or animation player are reused across the three very different applications. On the other hand, to maximize generality and reusability, developers should divide system features to small and simple components, but this will cause redundant communication among the components and may lower the system performance comparing to strait-forward an dedicated implementations.

Besides, we found the following problems in developing this framework. The description on the multi-modal input from the user is still quite trivial and can only capture simple actions done by the human user. We would like to strengthen this part to capture more complex conversational circumstances in the future. It was difficult to develop general purpose components for various applications, for example, to show subtext in the animator. Sometimes, there was problem in timing because we can not get direct control inside a model, for example, the TTS engine starts slowly in first run trial. The available action set is still small and can only be used with limited applications. Although GSML is meant to be used by system developers, writing GSML script may be tedious and error-prone, an authoring tool is desirable.

The ultimate goal of ECA and AI research is to develop a system with a high level of intelligence which can not be distinguished from a real human being. We would like to extend the framework to support the development of more complex deliberate process in the future.

[7]http://www.mindmakers.org/projects/SAIBA

6. ACKNOWLEDGMENTS

This research is partially supported by the Ministry of Education, Science, Sports and Culture of Japan, Grant-in-Aid for Scientific Research (S), 19100001, 2007, "Studies on Construction and Utilization of a Common Platform for Embodied Conversational Agent Research" and the Ministry of Science Education and Sports of the Republic of Croatia, grant nr. 036-0362027-2028 "Embodied Conversational Agents for Services in Networked and Mobile Environments."

7. REFERENCES

[1] C. Becker, S. Kopp, and I. Wachsmuth. Simulating the emotion dynamics of a multimodal conversational agent. In *Proceedings on Tutorial and Research Workshop on Affective Dialogue Systems (ADS-04)*, 2004.

[2] J. Blom and A. Monk. One-to-one e-commerce: who's the one? In *the Proceedings of CHI 2001*, pages 341–342, 2001.

[3] J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjlmsson, and H. Yan. Embodiment in conversational interfaces: Rea. In *The Proceedings of CHI99*, 1999.

[4] J. Gratch, J. Rickel, E. Andre, J. Cassell, E. Petajan, and N. Badler. Creating interactive virtual humans: Some assembly required. *IEEE Intelligent Systems*, 17(4):54–63, 2002.

[5] M. Klesen, M. Kipp, P. Gebhard, and T. Rist. Staging exhibitions: methods and tools for modelling narrative structure to produce interactive performances with virtual actors. *Virtual Reality*, 7(1):17–29, December 2003.

[6] S. Kopp, B. Krenn, S. Marsella, A. N. Marshall, C. Pelachaud, H. Pirker, K. R. Thorisson, and H. Vilhjalmsson. Towards a common framework for multimodal generation: The behavior markup language. In *The Proceedings of the 6th International Conference in Itelligent Virtual Agents (IVA2006)*, pages 205–217, 2006.

[7] Y. Nakano, M. Okamoto, D. Kawahara, Q. Li, and T. Nishida. Converting text into agent animations: Assigning gestures to text. In *Proceedings of The Human Language Technology Conference (HLT-NAACL04)*, 2004.

[8] K. Oka and Y. Sato. Real-time modeling of a face deformation for 3d head pose estimation. In *Proc. IEEE International Workshop on Analysis and Modeling of Faces and Gestures (AMFG2005)*, 2005.

[9] H. Vilhjálmsson, N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. N. Marshall, C. Pelachaud, Z. Ruttkay, K. R. Thórisson, H. van Welbergen, and R. J. van der Werf. The behavior markup language: Recent developments and challenges. In *Proceedings of the 7th International Conference on Intelligent Virtual Agents (IVA2007)*, volume 4722 of *LNAI*, pages 99–111. Springer, 2007.