An Agent Based Multicultural User Interface in a Customer Service Application

Hung-Hsuan Huang[1], Aleksandra Cerekovic[2], Kateryna Tarasenko[1], Vjekoslav Levacic[2], Goranka Zoric[2], Margus Treumuth[4], Igor S. Pandzic[2], Yukiko Nakano[3], and Toyoaki Nishida[1]

[1]Graduate School of Informatics, Kyoto University, Japan, [2]Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, [3]Department of Computer, Information and Communication Sciences, Tokyo University of Agriculture & Technology, Japan, [4]Institute of Computer Science, University of Tartu, Estonia

Abstract—**The advancement of traffic and computer networks makes the world more and more internationalized and increases the frequency of communications between people using different languages and expressing different nonverbal behaviors. To improve communication of embodied conversational agent (ECA) systems with their human users, the importance of their capability to cover the cultural differences emerged. Various excellent ECA systems are developed and proposed previously, however, the cross-culture communication issues are seldom addressed by researchers. This project aims to explore the possibility of rapidly building multicultural and multimodal ECA inter-faces for customer service applications with a generic framework connecting their functional blocks.**

***Index Terms*— embodied conversational agent, distributed system, blackboard, user interface, non-verbal interaction**

I. Project Background

EMBODIED conversational agents (ECA) are computer generated humanlike characters that interact with human users in face-to-face conversation and possess the following abilities:

- Recognize and respond to verbal and nonverbal input
- Generate verbal and nonverbal output
- Perform conversational functions (e.g. utterance turn taking, feedback and repair mechanisms)
- Give signals that indicate the state of conversations as well as to contribute new propositions

To achieve these features, system assemblies such as natural language processing, sensor signal processing, verbal and nonverbal behavior understanding, facial expression recognition, dialogue management, personality modeling, emotional modeling, natural language generation, facial expression generation, gesture generation, and CG animator are required. These functions actually involve multiple disciplines like A.I., computer graphics, cognitive science, sociology, linguistics, psychology, etc. They are in so broad range of research disciplines such that virtually no single research group can cover all aspects of a fully operating ECA system. Moreover, the software developed from individual research result is usually not meant to cooperate with each other and is designed for different purpose. Hence, if there is a common and generic backbone framework that connects a set of reusable modulized ECA software components, the rapid building of ECA systems will become possible and the redundant efforts and resource uses of ECA researches can be prevented. For these reasons, our group is developing such a generic ECA platform and researching the adequate communicative interfaces between ECA software blocks. As a result, a basic system model is developed with a prototype system and described in the next section.

On the other hand, the advancement of traffic and computer networks makes the world more and more internationalized and increases the frequency of communications between people using different languages and expressing different nonverbal behaviors. To improve the communication of ECA systems with their human users, the importance of their capability to cover the cultural differences emerged. Although various excellent agent interface systems are developed and proposed previously, the cross-culture communication issues are seldom addressed by researchers.

II. Project Objectives

To explore the issues that may occur in multicultural communication, especially nonverbal communicative behaviors performed spontaneously by humans; we propose this project with the objective to develop a customer service application with an ECA interface which serves human users from different cultures based on the generic ECA framework. Based on the discussion among the team members prior to the workshop, the target application is decided to be a tour guide agent of Dubrovnik city where is specified as a UNESCO Worlds Heritage. Since most of the team members come from Japan and Croatia, it is most convenient to gather first-hand Japanese and Croatian cultural information where the differences are supposed to be fairly obvious. A guide agent dynamically changes its behaviors either in Japanese way or in Croatian way according to its

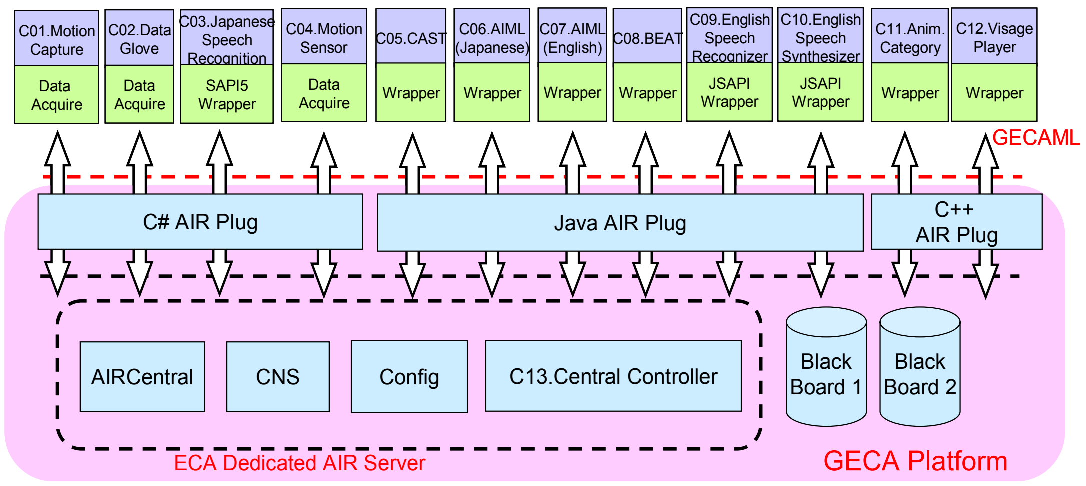

Fig. 1 The conceptual diagram of GECA framework and the configuration of the eNTERFACE06 agent

visitor was thus suggested. On the other hand, because of the lack of good-quality speech synthesizer/recognizer for Croatian, the guide agent will speak and listen to English in its Croatian mode.

In this system, the agent mediates the site seeing information of Dubrovnik to its visitors via verbal and non-verbal interactions. An example scenario is: when a visitor comes to the system, the system recognizes the visitor as a Japanese or Croatian from the combination of the speech recognizer's result and the non-verbal behaviors of the visitor such as bowing in greeting in Japanese culture. The agent then switches to its Japanese mode, that is, speaks Japanese and behaves like a Japanese to accept and answer the queries from the visitor while performing culture-dependent gestures according to predefined scenarios in that session. At the same time, the visitor can interact with the agent not only by natural language speaking but also by non-verbal behaviors such as pointing to an object on the background image or raising his (her) hand to indicate that he (she) wants to ask a question. Besides, to reduce the system complexity and prevent the drawbacks come from an ill-implemented 3D environment, in the prototype system that is going to be implemented during eNTERFACE'06, scene transitions are approximated by camerawork and the changes of realistic background photos instead of building a full 3D virtual world.

III. Generic ECA Framework

To connect many heterogeneous functional components to an integral virtual human, the consistency of all communication channels and the timing synchronization of all components will be very important issues. Also, to handle nonverbal inputs from humans, the capability to handle streaming data from sensors in real-time is indispensable. Our platform is built upon a routing and communication protocol of cooperating A.I. programs, OpenAIR [1]. The platform mediates the information exchange of ECA software components with XML messages via shared memory mechanism (blackboard or white boards in OpenAIR's context) and will have the following advantages:

- Distributed computing model over network eases the integration of legacy systems
- Communication via XML messages eliminates the dependency on operating systems and programming languages
- Simple protocol using light weight messages reduces the computing and network traffic overhead
- Prioritized messages make quality of service control possible and facilitates real-time event processing (not implemented yet)
- Explicit timing management mechanism (partially implemented)
- Support discrete messages and streaming sensor data at the same time (partially implemented)
- The use of shared backbone blackboards flatten the component hierarchy, shorten the decision making path and can realize reflexive behaviors
- Possible to use multiple logically isolated blackboards rather than traditional single blackboard (not implemented yet)
- Components can communicate with each other directly or via blackboard(s) (not implemented yet)
- Easy to switch or replace components which have the same function if they understand and generate messages in the same type

Figure 1 shows the conceptual diagram of the GECA framework and the configuration of the planed Dubrovnik tour guide agent. Based on this framework, we are specifying an XML based high-level protocol for the data exchanges between the components plugged into the GECA platform. Every GECA message belongs to a message type, for example, "input.speech.text", "output.action.speak", etc. Each message type has a specified set of XML elements and attributes, for example, "intensity", "duration", "start_time", etc. The message flow works like the following scenario upon the platform, when a component starts; it registers its contact information (unique name, IP address, etc) to *CNS* (Central Naming Service) component and subscribes its interested message type(s) to the *AIRCentral* component. Then the messages in those types will be sent to the component from the specified blackboard (or a *whiteboard* in OpenAIR's terminology) which behaves like a shared memory between the components when some other component *published* the messages. That component then processes the data it got and publishes its own output to the shared blackboard in certain message type.

By utilizing the communicative functions provided by the Air Plug libraries (currently we have developed the C#, C++ version libraries and a customized Java reference implementation from mindmakers.org) which are a part of the platform, an ECA system builder needs to develop a small piece program called a *wrapper* in order to handle and convert the input/output of an existing software component to be GECAML (Generic ECA Markup Language) compliant. After doing this, the heterogeneous nature of components that provide the same capability (for example, both of a MS SAPI4 TTS and a JSAPI TTS provide the same capability of the agent, i.e. to speak out from text) can be hided and behave identically to the other software components.

IV. Dubrovnik Tour Guide Agent

During the eNTERFACE06 project's four-week period, the participants of this project cooperated to develop the tour guide agent described in section II. In this section, we discuss the GECA software components that are used in this project and the main tasks that were dealt during the project period.

A. Software Component Configuration

This agent was planned with the component configuration depicted in Figure 1. The follows are the brief descriptions of those software components.

C01. Motion capture component. This component utilizes a simple motion capture device [2] using IR technology to roughly approximate a predefined set of human visitor's non-verbal behaviors.

C02. Data glove component. This component acquires data from a data glove hardware device and reports recognized movements of the visitor's fingers to the other components.

C03. Japanese speech recognition component. This component is a wrapped SAPI-5 Japanese recognition engine, Julius [3] and has been implemented.

C04. Motion sensor component. This component acquires data from a 3 dimensional acceleration sensor [4] which is attached on the visitor's head to detect head shaking and nod movements. This component has been implemented.

C05. Japanese spontaneous gesture generating component. This component is a wrapper of CAST [5] engine which generates the type and timing information of spontaneous gestures from Japanese utterance input string. This component has been implemented.

C06. AIML interpreter components for Japanese. This component wraps a Java implementation [6] of AIML [7] interpreter. It reads one or more AIML scripts which specify the agent's verbal and nonverbal responses to certain input behaviors from the visitors. Therefore, this component behaves like the brain of the agent and thus the current agent shows only reflexive behaviors with some context referencing capability comes with AIML and has no internal state. Besides, because the original AIML does not accept customized tags, a set of tags specifying visitor's non-verbal inputs and agent's non-verbal outputs must be encoded into the script. The wrapper of this component has been implemented but the scenario script(s) has to been defined during the eNTERFACE workshop.

C07. AIML interpreter components for English. The same as above except this component handles English inputs / outputs.

C08. English spontaneous gesture generating component. This component is a wrapper of BEAT [8] which generates the type and timing information of spontaneous gestures from English utterance input string. This component has not been implemented yet.

C09. English speech recognition component. This component wraps a speech recognition engine to recognize English speaking of the visitor and from predefined grammar rule and sends the recognized result as a text string to the subscribed components. This component has not been completed yet.

C10. English Text-To-Speech component. This component wraps an English Text-To-Speech (TTS) engine to generate the voice output of the agent and visime events to drive the character animator to move the agent's lips. This component has not been completed yet.

C11. Animation category component. This component is a database storing the number values of MPEG4 FBA parameters of a predefined set of animation / action to drive the character animation in real-time. This component has not been implemented yet.

C12. Character animation player component. This component is a wrapped character animation player which is implemented in visage|SDK [9]. It accepts driving event messages from the animation category and speech synthesizer component and performs the specified character animation.

C13. Central controlling component dedicated to ECA. This component is one part of the OpenAIR server and handles synchronization among the components, ensures integrity of all output modals, selects the actions to perform if there is some contradiction.A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

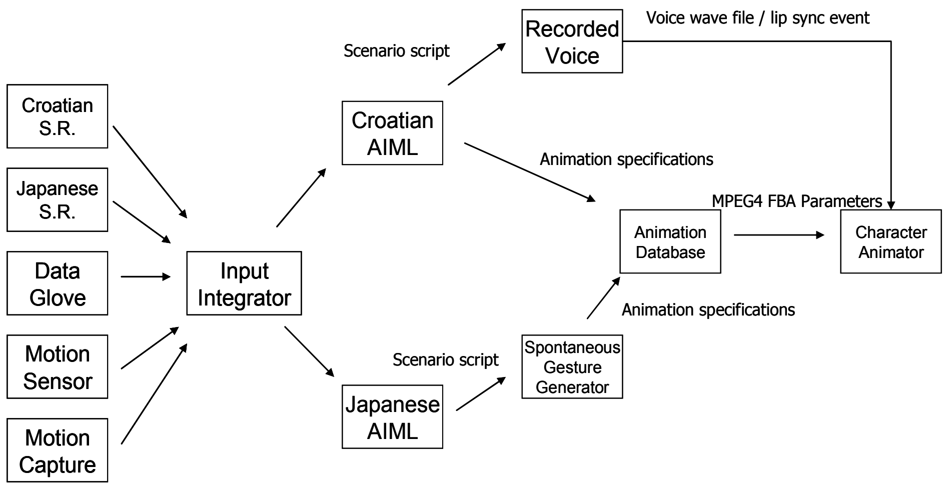

Fig. 2 The data flow of the Dubrovnik tour guide agent

However, during the development period, there were some modifications made to the original plan. Because the venue of the workshop is in Croatia, we decided to use pre-recorded Croatian voice instead of English TTS and use English speech recognition engine to recognize a limited range of Croatian vocabularies. Figure 2 shows the data flow among the components of the actually built Dubrovnik tour guide agent. The speech recognition component and sensor components gather and recognize the verbal and nonverbal inputs from the human user and send the results to the AIML component. The inputs from different modals are combined by the wrapper of AIML component and are matched with predefined scenario AIML scripts. The AIML then sends the matched response which may include utterance and action tags to speech synthesizer and animation category component. Depends on the design of the spontaneous gesture generating component, the speech synthesizer component may output the generated voice by itself and send the visime events to the animator to drive the agent's lips or leave these jobs to the other components. In either way, timing information is sent to the spontaneous gesture generator. The spontaneous gesture generator inserts action tags into the utterance according to the timing information from the speech synthesizer and its natural language tagging companion. The animation category listens to action queries from the spontaneous gesture generating component or the AIML component and sends FBA (Facial Body Animation) parameters to drive the character animator. The character animator listens to action and visime events and play them in real-time. Some character animators (e.g. visage) may also provide TTS support; in that case, it also listens to the utterance output of the spontaneous gesture generating component. Furthermore, shortcuts between the sensor components and the animator that bypass the pipeline are allowed and make reflexive behaviors of the agent possible, and this is one of the strengths of this framework over the other ECA architectures.

B. Non-verbal input recognition

To provide an immersive environment for the user to interact with the tour guide agent, a LCD projector with tilt-compensation function is used to project a large enough image of the agent on the screen. The user then stands in front of the screen and interact with the guide agent as (s)he is really in the virtual Dubrovnik space.

In the non-verbal input recognition issue, the aim is to detect the following behaviors from the user:

- Get the agent's attention
- Point to the interested objects shown on the display
- Show the willing to ask a question
- Interrupt the agent's utterance
- Shake head and nod to express positive and negative answers

Because of the nature of the eNTERFACE workshop, only small size and portable sensor devices are adopted in this project. These non-verbal behaviors are recognized by using the data from data gloves, infrared camera, and acceleration sensors.

Nissho Electronics Super Glove

This data glove is a simple wearable input device which user can put on his right hand to detect finger curvature. Ten sensors, two for each finger, are used to detect how fingers are bent. Prior to first use, user must calibrate the glove's sensor readings by putting the fingers into three different positions. Data glove is connected with a cable to the control box which is a power input device and a processing unit of the data collected from the sensors. Control box can be connected to the PC with a serial cable and a serial port reader can be used to read the glove data. Data from the glove is represented with thirty ASCII characters. Three ASCII characters are assigned to each sensor where "000" means that a finger is straight and "900" means that it is fully curved. In a program we developed, we assign a threshold value of when finger becomes bent, which means that we detect only two states of the finger. By mapping finger shapes into the gestures it is easy to detect different kinds of positions like pointing or five fingers straight.

NaturalPoint OptiTrack FLEX3

Infrared reflecting materials are used to help detecting the approximate pointing direction of the user's right hand. Material in a shape of a strap is put around the wrist of the right hand and its spatial coordinates are detected by the OptiTrack infrared camera and projected into the 2D plane.

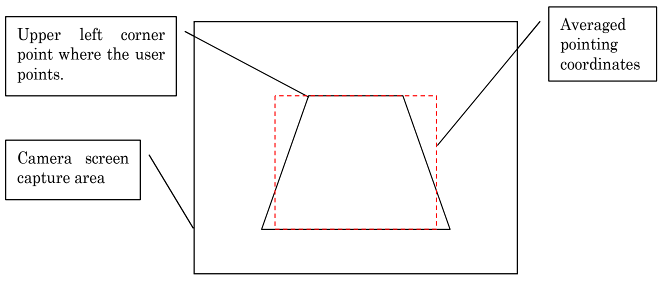
Upper left corner point where the user points.
Camera screen capture area
Averaged pointing coordinates

Fig. 3 The calibration of project 2D coordinates

During initial use, system must be calibrated. To calibrate the system, user stands in front of the projection screen and a camera and follows the software instructions to point to the each corner of the projection screen. Projected coordinates of the upper corners are bent inward and give overall projection shape resembling the trapezoid because camera stands closer to the floor. The concept is shown in figure 3. That inevitably leads to the curved projection of the hand movement. Nevertheless, it is assumed that interesting scenario objects will have perceivable size compared to the projection screen and that approximate pointing coordinates should be detectable.

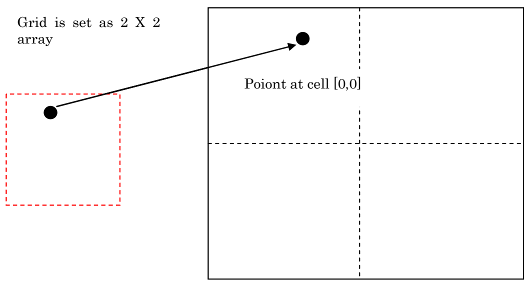

Fig. 4 Mapping between raw data and application coordinates

After the trapezoid's corner points are calculated we approximate the area of hand movement with a rectangle by averaging trapezoid's neighbor values. The projection screen is divided into the grid of arbitrary size. When user is pointing to the screen, his pointing coordinates are inside one of the grid's cell. Interesting scenario objects are mapped to distinct cells and therefore pointing at scenario object can be easily detected. This concept is shown in figure 4.

Camera's API gives us the information of where the detected object's center of mass is. If multiple infrared sources exist, more than one center of mass will be detected. This can especially occur when marker, due to the various reasons, has interruption in continuity of its area. To remove that problem, we use K-Means algorithm to group close centers. Software also removes all centers which show spatial stability in time. Such an example is a LCD projector which is put in front of the camera and disturbs the detection by constantly emitting the infrared light.

Software detects hand stability and waving. To detect if user holds his hand in a stable position, we need a time buffer of marker coordinates. Buffer size is determined by the camera's frame rate. After each frame algorithm calculates the central point of all points in a buffer and distance from that central point to each other point in a buffer. If all distances are below the predefined percent of screen width, hand stability is alerted.

Swing is defined as a hand movement from one point to another, before hand changes movement direction. Software detects waving which goes from the elbow to the forearm, where the marker's waving is visible on the screen and recognizable by the system. Waving pattern is characterized with two specific features. First one is that a length of each swing is approximately the same as a previous one. Also, we tend to wave in a constant speed without variations in a hand movement. Therefore, gradient of a hand movement and movement length seem like a suitable features to detect the waving. To detect the hand movement, buffer is filled with the waving data. Each change in direction of the marker that is larger than some threshold is detected as a new swing. Pivot element, first element in a buffer, is taken as a reference point to calculate if waving is occurring. By comparing other waving elements found in a buffer with a pivot element, specifically their gradient, length and number of swings, we may detect and signalize the waving.

NEC/Tokin 3D Motion Sensor

It is a sensor that can detect the change of acceleration in three dimensions. This small-size sensor is attached on the top of the headset that is usually used for gathering speech inputs, and the data from it is used to detect the movement of the user's head. A component that detects two types of head movements, nodding and shaking was developed. It generates output message to represent positive (nodding) and negative (shaking) verbal answers. Therefore, the user can nod instead of saying "yes" or shake his (her) head instead of saying "no."

Data glove and hand movement detection programs work as a separate .NET applications. Each program has an OpenAir plug implementation, sending the data which *InputManager* component receives and combines into the new output (see Table 1). *InputManager* component acts as a fusion component of the different input modalities, and is made as a simple state machine. It sends the output only when new gesture is detected.

TABLE 1
THE USE OF SENSOR DATA

Component name	finger_shape		
Outputs	**Type**	**Content**	**Description**
	in-put.shape.fingers	FiveUps	Five fingers straight
		Pointing	2 finger straight
		Victory	2,3 fingers straight
		Unknown	Any other combination
Component name	wrist_position		
Outputs	**Type**	**Content**	**Description**
	in-put.position. wrist	UNKNOWN[*X,Y* or OUTSIDE], where the X,Y is the cell where the marker is detected	
		WAVING_DETECTED[*X,Y* | OUTSIDE]	
		HAND_STABLE[*X,Y* | OUTSIDE]	
		CALIBRATION[*calibration message*]	Sends the message in which status the calibration is
Component name	input_manager		
Outputs	**Type**	**Content**	**Description**
	in-put.manager. fusion	POINTING[X,Y|OUTSIDE]	Pointing + HAND_STABLE + coordinates
		WAVING	FiveUps + WAVING_DETECTED
		ATTENTION	FiveUps or Victory + HAND_STABLE
		UNKNOWN	

C. *Croatian Speech Input / Output*

As there is no Croatian speech recognizer it is decided that we will use English speech recognizer to recognize Croatian. Therefore simple rule grammar had to be created to recognize some words according to the agent scenario. The grammar for Croatian is defined by using English alphabet to approximate the pronunciation of Croatian even those words do not exist in English. CloudGarden [10] library is used to access SAPI5 compliant Speech-Recognition engine by standard Java Speech SAPI.

It was possible to create grammar according to the scenario to make English speech recognizer to recognize Croatian. We have made such grammar to recognize both only Croatian words and whole sentences. Both ways were quiet successful. However, several problems came out. Some Croatian words were impossible to write with English alphabet, therefore it was better to avoid them and use some other words instead. Also, if grammar contained several very similar words, they were sometimes mixed by recognizer, so it is better to choose words that are not so similar (since scenario was not that strict this was possible). And the last thing, although the recognition worked with all tested subjects, recognition with some was slightly better.

Once we had Croatian scenario, a native Croatian speaker has recorded speech the agent was supposed to say in certain situation in the noise free room. By applying a lip sync application we have [11], which takes speech as input and gives animation (of the lips) as output, we have created animation from the prepared speech files.

Our automatic lip sync system determines the motion of the mouth and tongue during the speech by speech signal analysis. Neural networks are used to classify the speech into a sequence of visemes (visual representatives of phonemes). In order to obtain training data for the NNs, a training set with visemes was collected. The speech is first preprocessed. Input in NNs are MFCCs calculated from training data and output is different viseme classes. When correct viseme is chosen, it can be sent to animated face model. MPEG-4 standard is used for generating facial animation since facial animation can be generated for any parameterized face model if the visemes are known. The method is implemented in C++. The program reads speech from pre-recorded audio files and continuously performs spectral analysis of the speech. Suitable visemes are shown on the screen or saved in the FBA file.

At the end, we had a pair of speech-animation files for every situation according to the scenario.

D. *Action Animation Database*

By an animated action we mean a set of Face and Body animation parameters displayed within the same time interval to produce a visual agent action, such as nod or gesture. The queries for animated actions for the agent are stored in the AIML script. A single database query corresponds to an AIML category consisting of a *pattern* (typically, a human’s action) and a *template*, the agent’s reaction to the pattern. The description of an animation, which is to be started simultaneously with a particular part of the agent’s utterance, is incorporated in the <template> tag using the “[“ and ”]” characters.

Below is a simple example of an AIML category with non-verbal input/output descriptions:

```
<category>
<pattern>What is this
[PointingAt Object=“monastery”]
</pattern><template>This is the big Onofrio’s Fountain
[Action Type="pointing" SubType="Null" Duration="2300"
Intensity="0" X="0" Y="0" Z="0" Direction="rightUp"
ActivationFunction="sinusoidal"/] built in 15th
century. The Fountain is a part of the town's water
supply system which Onofrio managed to create by
bringing the water from the spring located 20 km away
from town.</template></category>
```

Here, the non-verbal action “pointing” of the agent character is described. Its duration is specified by opening tag and closing tags that enclose a segment of an utterance and thus the actual value depends on the TTS (Text-To-Speech) synthesizer if it supports prior phoneme timing output or absolute values in milliseconds. The attribute SubType has the value of “Null”, as there are no possible subtypes defined for it. The “Intensity” attribute is to have integer values, with “0” value meaning that the intensity is not specified for the action in question. Other actions, for which the attribute "intensity" has sense, do have an intensity scale specified. For example, to distinguish between a slight bow used while greeting in European countries from the deep Japanese salutation bow, we introduce a scale of values for the "bow" action.

Further, we envisage the use the coordinates ("X", "Y", "Z") integer-valued attributes in the future. The meaning of these coordinates will be dependent on the action. For example, for the "pointing" action such this triad would mean the position on the background screen where the agent is supposed to point. At the moment the alternate attribute "Direction" is used.

The "ActivationFunction" attribute stands for the dynamics of the action. Possible values are "linear", which uses a linear function to activate the corresponding MPEG4 Face and Body Animation parameters (FAPs), "sinusoidal", which uses trigonometric functions to activate the FAPs, and "oscillation" function, which is used for the repeated actions, such as "Nodding" or "HeadShaking". In addition to these attributes, the attribute "sync" with possible values "PauseSpeak", "BeforeNext", "WithNext" specifies the synchronization between non-verbal actions and speech synthesizer.

The action "pointing" is an invocation of one character action with the name "pointing" which is stored in a high-level action database. The database is currently implemented as one part of the visage animator and stores low-level MPEG4 FBA parameter specifications as well as run-time configurable parameters for high-level action invocations.

Visage operates with FBAPs – a large set face and body animation parameters, specified by MPEG4. FBAPs are divided into FAPs (Face animation parameters) and BAPs (body animation parameters). The BAP parameters are the angles of rotation of body joints connecting different body parts, such as toe, ankle, knee, hip, spine joints, shoulder, clavicle, elbow, wrist, and the hand fingers. There are 64 low-level FAPs, which are closely related to muscle actions and represent a complete set of basic facial actions, and two high-level FAPs (expression and viseme).

Example: by analyzing the videos, we found that for the pointing gesture can be animated by manipulating such FAPs as head_yaw and head_pitch, and the following BAPs: l_shoulder_flexion, l_shoulder_abduct, l_shoulder_twisting, l_elbow_flexion. We ran a series of experiments to set the values of these parameters.

The analysis of videos to create gestures was pretty difficult because we did not use any tool that would translate a gesture in to a set of FBAPs. Without it took us from 5 to 30 experiments, depending on the complexity of the action, to adjust the parameters for an action. Needless to say, that such approach is rather time-consuming.

Then, for the most of the actions implemented in our system, we divide the duration of it into 3 states: attack, sustain and decay. For each of the intervals, depending on the activation function of the action in question, we define how the parameter value changes as a function of time. For example, in case of sinusoidal function (as is with the pointing gesture), in the attack phase, the value of the parameters changes as a sinusoidal (increasing function) function of time, whereas in the sustain (i.e. peak) phase it changes as a constant. Finally, in the decay phase the corresponding function is cosinusoidal (decreasing). The character animation created for this application is listed in Table 2.

Cultural differences through the non-verbal behavior of the agent

We observed analyzed non-verbal behaviors of Japanese tour guides and took videos. As a result, we tried to implement these features in our agent. Also, some very typical emblem Japanese gestures, that are not inherent to the European culture, were implemented. For example, the so-called "handsCrossed" gesture. This gesture seems to be pretty unique, and normally draws attention of Western people who first come to Japan and are used to head shaking or simply verbal expression of prohibition (See Figure 5 and Figure 6). In Japanese culture, to show that something is prohibited, people tend to cross their hands before the chest. Sometimes, this action is accompanied with head shaking. Similarly, our agent uses this gesture when prohibiting in the Japanese mode, in contrast to the European mode, where only head shaking is envisaged.

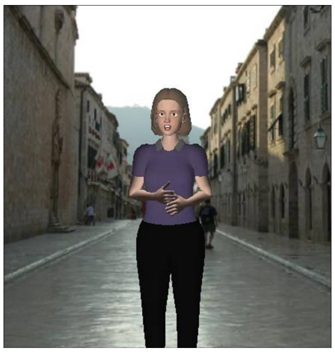
Fig. 5 A Japanese emblem gesture to show prohibition

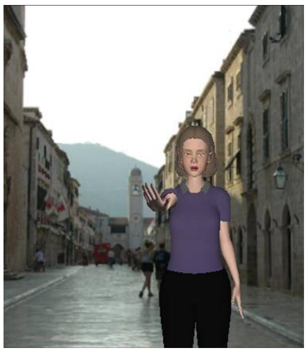
Fig. 6 Another example is the "Negation" gesture in the Japanese mode: waving with a hand while the arm is extended. In Japan, the negation is expressed by shaking one's upright hand near one's mouth with two thumbs closer to one's face. Sometimes shaking head sideways is also added. When asking to wait, Japanese people usually show the palm of one hand to another person. At times, both hand maybe used.

TABLE 2
CHARACTER ACTION ANIMATION CREATED FOR THE DUBROVNIK GUIDE APPLICATION

Gesture/attributes	Values	Meaning
Pointing: - Direction	left, leftUp, right, rightUp, rightForward, leftForward, backH, backE	The agent points in the directions that semantically correspond to the values defined for this attribute. In future, we plan to use the coordinates on the screen to which the agent should point. Thus using direction instead of the coordinates is a temporarily solution. "backE" and "backH" values represent variations of gestures with the elbow bent.
Bow -Intensity	1-3	1 corresponds to a shallow bow, using only head; 2- is a deeper bow, very frequently used by Japanese people in a daily conversations, 3-corresponds to a very polite bow, showing a high respect to the listener
Invite -Subtype	Croatian, Japanese	The "invite" action of the "Croatian" subtype is waving upwards and then backwards with the left hand, a somewhat informal emblem gesture meaning inviting. The action of the subtype "Japanese" has not been implemented yet.
HandsCrossed		This is an emblem Japanese gesture, meaning that something is not allowed. The hands are crossed in front of the lower part of the chest
Nodding		The action meaning both in Croatian and Japanese agreement, consent.
ShakeHead		The action meaning both in Croatian and Japanese negation or disapproval.
Extend	[at the moment, "extend" means extending the right arm. In the future we might need extending the left arm as well. Thus, the subtype attribute might be introduced with the "left/right" as possible values.]	This action means right arm extended with the palm open and oriented upwards. The meaning in the Japanese culture is "wait please"
Wave	[at the moment, "wave" means waving with the right hand. In the future we might need waving with the left hand as well. Thus the subtype attribute might be introduced with the "left/right" as possible values.]	This action means oscillating right hand waving. Used in combination with the "extend" action as part of the Japanese gesture meaning "No. This is not true".
Expression -Subtype	"smile"	This value corresponds facial expression "joy" defined in the Class SimpleFacialExpression
Walk -Direction	"right", "left", "back"	The agent walks in the directions that semantically correspond to the values defined for this attribute. In future, we plan to use the coordinates on the screen to which the agent should walk. Thus using direction instead of the coordinates is a temporarily solution.
Beat -Subtype	"a" –"e"	Waving spontaneous gestures with either one or both arms, used by the BEAT engine.
Contrast -Subtype	"a" –"c"	Waving spontaneous gestures with either one or both arms, used by the BEAT engine.NOT IMPLEMENTED YET
Warning		An emblem gesture meaning danger : the elbow bent and the hand raised. In future, the finger feature needs to be implemented, i.e. the pointing finger only pointing upwards.

In addition to the character action specification tags, animator controlling tags such as "Scene" are also defined. This tag informs the animator to switch scene settings while the scenario advances. Besides, the "PointingAt" tag is generated by a component which maps raw coordinate data to object names according to the information provided by motion capture device and scene changing messages.

E. Character Animator

During the workshop, the main improvement upon the character animator is the adoption of ARToolkit [12] to align the position of the character with the background images.

The ARToolKit 2.65 video tracking libraries capture real time input from real camera and detect presence of the marker in the input picture. If marker is detected, real camera position and orientation relative to physical markers in real time are calculated. Calculated parameters are used for rendering a virtual object on physical marker.

The main idea is to use ARToolkit libraries in one separate application that will recognize marker on the static picture (background picture). Marker will define the a position of the agent character on the picture. As output, this application will give calculated parameters that will be used in Visage player for rendering the agent character aligned with a same background, but without marker on the picture. In order to realize this, two pictures of the same background had to be taken, one with and another without marker.

At first, some modifications to existing *Simple* demo application, created by ARToolkit developers, had to be made. *Simple* application is programmed in Visual Studio 6.0 in C program language. Application is considering video stream in input and during continuous reading of sequential pictures from frames it is recognizing marker in each picture, if present. Markers on the picture are distinguished by pattern objects that are previously saved by another application, called *Make pattern*. If the marker is recognized, parameters of the modelview matrix and projection matrix needed for drawing virtual object on a marker are calculated for every picture. After that, function for drawing virtual object on physical marker is called. As result of *Simple* application, an output window where input picture align with virtual object on the marker, if present, is displayed.

In order to read a static picture from specified location instead of real time input from camera, eight functions that are receiving video input in ARToolkit are changed to return NULL value. Size of the display window that was calculated automatically by video functions is increased by decreasing zoom value while displaying main window. As size of a picture, a constant value 1250x 937 is set. For reading static pictures into unsigned bytes is used Sourceforge DevIL library.

After these modifications several tests of a *Simple* application with various .jpg pictures are done. Pictures are made by camera and are differed in the position and rotation of the marker. Marker is put on the floor with a different distance relative to the camera and also on top of the tripod. These pictures are then used as input value to *Simple* application in order to check percentage of detection. When application is started, specified picture is read into unsigned bytes. After that, read bytes are checked if specified marker (saved by *Make pattern*) is present. If marker is detected, as output of modified *Simple* application, static picture with virtual object on the marker is displayed. In the same time, calculated parameters are saved in .txt file.

As a result of the tests, in output window only 20% of markers on various pictures are detected. Results of these tests were not satisfying.

In the all pictures that were detected in previous test, markers were put in the front of the camera on the floor and were lightened. Pictures with marker on the tripod had also good results, but they were not considered in later work because of the susceptibility of a marker position. If marker is slightly moved on the tripod, virtual object that is rendered on the picture is moved. Besides that, marker attached on the tripod was slightly rotated to the floor, so local coordinate system of a virtual object wasn't aligned to the floor. Conclusion of this test is to use a bigger marker and to lay it on the floor without presence of any shadow on the marker.

Further, different results of detection were also noticed in the two other things. In *Simple* application threshold value of an input picture can be changed manually. In some of the pictures that had bad results, marker was detected for changed threshold values. Second, new *Make pattern* application is made. Original ARToolkit version of the *Make pattern* application was used to save marker pattern from picture captured by USB camera. New *Make pattern* application that is created can detect and save presence of marker from a static picture. Therefore, two different pattern objects for one marker can be used as input of *Simple* application.

Input value of final application is set of picture's names to be processed. After one picture is being read, presence of the marker on the picture is checked for different threshold values and both patterns made by *Make Pattern* application. This parameters are changed automatically in the application until marker on the picture is recognized. Output of the application is display window with static picture and virtual object and text file with parameters of the modelview matrix and projection matrix for each input picture. Output text file can be used by Visage player for positioning the agent character depending of the loaded background picture.

Output values, parameters of final background image alignment application give very good results in alignment of the agent character to the background image in Visage Player. However, this result can be used for only one picture separately. Later improvements of this system can include continuous scene transitions. The main idea is to generate continuous scenes while agent is walking, e.g. walking around circular fountain. In order to do realize this idea, detection of two different markers on the set of static pictures has to be included in *Simple* application. Set of static pictures has to be made while moving camera and continuously changing position of one marker after another, like character is walking. After this, output values of application will be two pairs of parameters for each picture. Later, these parameters can be used in Visage Player to calculate position of virtual character while moving from one picture to another.

V. Project Outcome and Conclusion

To the end of the workshop, we could not manage to achieve all of the planned project objectives. The individual non-verbal input components and Croatian speech contents are completed but are not integrated into the system. The final demonstration was done with a Dubrovnik guide agent running in two modes, English and Japanese modes. In Japanese mode, since there is a spontaneous gesture generating component, the agent's presentation looks more natural because the beating gestures performed at the timing generated by CAST engine. On the other hand, in English mode, the agent performs scripted gestures only.

However, this is our first time to apply the GECA framework to a larger and non-trivial testing application. We got some valuable experiments in component development and message specifications. Automatic character-background alignment application is developed and a suitable parameter set for dynamically configurable character animation is explored.

References

[1] OpenAIR protocol, http://www.mindmakers.org/openair/airPage.jsp

[2] NaturalPoint OptiTrack Flex 3, http://www.naturalpoint.com/optitrack/

[3] Julius Japanese speech recognition engine, http://julius.sourceforge.jp/en/julius.html

[4] NEC/Tokin 3D motion sensor, http://www.nec-tokin.com/english/product/3d/index.html

[5] Nakano, Y., Okamoto, M., Kawahara, D., Li Q., Nishida, T.: Converting Text into Agent Animations: Assigning Gestures to Text, in *The Proceedings of The Human Language Technology Conference* (HLT-NAACL04), 2004.
[6] Program D, http://www.aitools.org/Program_D
[7] AIML (Artificial Intelligence Markup Language), http://www.alicebot.org/
[8] Cassell, J., Vilhjalmsson, H., Bickmore, T.: BEAT: the Behavior Expression Animation Toolkit, in The Proceedings of SIGGRAPH '01, pp.477-486, 2001.
[9] visage|SDK, visage technologies, http://www.visagetechnologies.com/index.html
[10] TalkingJava SDK, CloudGarden.com, http://www.cloudgarden.com/index.html
[11] Zoric, G., Pandzic, I.S.: A Real-time Language Independent Lip Synchronization Method Using a Genetic Algorithm, in the proceeding of ICME 2005, 6-8 July 2005.
[12] ARToolKit, http://artoolkit.sourceforge.net/

Principal investigator:
Hung-Hsuan Huang graduated from the Computer Science Department of National Chen-Chi University, Taiwan in 1998 and obtained his master degree of computer science and information engineering from National Taiwan University, Taiwan in 2000. After a two-year military service where he was the political warfare director and the co-commander of an army company, he came to Japan. After the learning in a Japanese language school for one year, he entered the Ph.D. course of the Graduate School of Informatics of Kyoto University, Japan in 2003. His research interests include intelligent software agent, information visualization, photo management, gesture interface and is working on the generic ECA platform topic.

Project Advisors:
Prof. Toyoaki Nishida received the B.E., the M.E., and the Doctor of Engineering degrees from Kyoto University in 1977, 1979, and 1984 respectively. In 1980, he joined Department of Information Science, Kyoto University as an Assistant Professor. In 1988, he was promoted as an associate professor. In 1993, he joined Graduate School of Information Science Nara Institute of Science and Technology as Professor. During 1998-2003, he led the Breakthrough 21 Nishida Project, which is a five-year project sponsored by Ministry of Posts and Telecommunications, Japan. In 1999, he moved to the University of Tokyo as Professor. In 2004, he moved to the current position at Kyoto University. His research area covers artificial intelligence in general. He has been working on natural language understanding, spatial reasoning and qualitative reasoning. His current research focusing on knowledge communication, including conversational knowledge process, knowledge sharing, and qualitative reasoning.

Asst. Prof. Igor S. Pandzic received his BSc degree in Electrical Engineering from the University of Zagreb in 1993, and MSc degrees from the Swiss Federal Institute of Technology (EPFL) and the University of Geneva in 1994 and 1995, respectively. He obtained his PhD from MIRALab, University of Geneva, Switzerland in 1998. In the same year he worked as a visiting scientist at AT&T Labs, USA. In 2001-2002 Igor was a visiting scientist in the Image Coding Group at the University of Linköping, Sweden. He is now an Assistant Professor at the Department of Telecommunications, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. His main research interests are in the field of computer graphics and virtual environments, with particular focus on facial animation, embodied conversational agents, and their applications in networked and mobile environments. Igor also worked on networked collaborative virtual environments, computer generated film production and parallel computing. Igor was one of the key contributors to the Facial Animation specification in the MPEG-4 International Standard for which he received an ISO Certificate of Appreciation in 2000.

Asst. Prof. Yukiko Nakano received her bachelor degree in psychology from Tokyo Women's Christian University, Japan in 1988, one master degree in educational psychology from the University of Tokyo and the other one in media arts and sciences from Massachusetts Institute of Technology, USA in 1990 and 2002, respectively. She obtained her Ph.D. in information science and technology from the University of Tokyo, Japan in 2005. Yukiko was a researcher of NTT Research Laboratories from 1990 to 2000 and was a sub-leader researcher of Research Institute of Science and Technology for Society from 2002 to 2005. She moved to the Department of Computer, Information and Communication Sciences of Tokyo University of Agriculture and Technology as an associate professor in 2005. With her special interest in Embodied Conversational Agents (ECA), she has been studying human face-to-face communication in psychology and communication science, and creating multimodal conversational interfaces based on a model of human communication behaviors.

Team Members:
Kateryna Tarasenko graduated from the National Technical University of Ukraine "Kiev Polytechnic Institute" with a master degree in "Intelligence Systems for Information processing and Decision making". She entered the Graduate School of Informatics of Kyoto University, Japan as a research student in 2005. Her research interests include: embodied conversational agents, simulation of non-verbal communication behaviors.

Goranka Zoric received her master degree from the Faculty of Electrical Engineering and Computing, the University of Zagreb, Croatia in 2005 and is currently both a PhD student and a research associative there. Her main interest is in the field of facial animation and its application with Internet and mobile technologies and virtual environments with particular focus on automatic lip synchronization and gesturing of synthetic 3D avatars based only on the speech input.

Vjekoslav Levacic graduated from the Faculty of Electrical Engineering and Computing of University of Zagreb, Croatia in 2005 and is currently a graduate student of the same university. His research interests include multimedia, software architecture, web design, image processing, HCI and mobile networks. (Vjekoslav will stay in Dubrovnik only for the first two weeks)

Aleksandra Cerekovic is in her fifth and final year as an undergraduate student of the Faculty of Electrical Engineering and Computing of University of Zagreb, Croatia. She worked on the topic of lip synchronization for real-time speech recognition and synthesis. Besides that, she has the research interests on the topics of embodied conversational agent and virtual environment.

Margus Treumuth received his bachelor and master degrees in computer science from University of Tartu, Estonia, in 2002 and 2004 respectively. He is now a PhD student in University of Tartu and has research interests in computational linguistics and dialogue systems. (Margus will leave on 22/07)