First Prototype of Conversational H.C. Andersen

Niels Ole Bernsen, Marcela Charfuelàn, Andrea Corradini, Laila Dybkjær, Thomas Hansen, Svend Kiilerich, Mykola Kolodnytsky, Dmytro Kupkin and Manish Mehta
NISLab, University of Southern Denmark, Campusvej 55, 5230 Odense M, +45 65 50 35 51
{nob, marcela, andrea, laila, thomas, kiil, mykola, dima, manish}@nis.sdu.dk

ABSTRACT

This paper describes the implemented first prototype of a domain-oriented, conversational edutainment system which allows users to interact via speech and 2D gesture input with life-like animated fairy-tale author Hans Christian Andersen.

Categories and Subject Descriptors

H5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – *animations, virtual realities, audio input/output*. H5.2 [**Information Interfaces and Presentation**]: User Interfaces – *natural language, voice I/O, interaction styles, input devices and strategies*.

General Terms

Documentation, Performance, Design, Human Factors.

Keywords

Domain-oriented spoken conversation, life-like animated agents.

1. INTRODUCTION

This paper presents work carried out in the European NICE project (2002-2005) on Natural Interactive Communication for Edutainment [7]. The main goal of the project is to demonstrate natural human-system interaction, in particular involving children and adolescents, by developing natural, fun and experientially rich communication between humans and embodied historical and literary characters, including H. C. Andersen and some of his fairy tale characters. We describe the implemented first prototype (PT1) of the NICE Hans Christian Andersen (HCA) system in which we aim to demonstrate domain-oriented conversation, including 2D gesture input, with life-like animated fairy-tale author HCA. The work on his fairy tale characters is not addressed.

By contrast with task-oriented spoken dialogue [2], domain-oriented conversation has no task constraints. The user can address, in any order, any topic within HCA's knowledge domains, using spontaneous speech and mixed-initiative dialogue. In PT1, the domains are: HCA's fairy tales, his life, his physical presence in his study, the user, and his role as "gate-keeper" for access to the fairy tale world. In addition, HCA has a 'meta' domain in order to be able to handle meta-communication during conversation. HCA reacts emotionally to the user's input, e.g. by getting angry or sad or becoming happy if the user likes to talk about his fairy tales. The NICE HCA system is *not* an information system. It attains its educational goal by providing correct factual information, visually and orally. An equally important goal is to entertain through conversation, to make the target users of 10-18 years old kids and teenagers pleased to meet someone of, and from, a different age who is much more like themselves than expected.

Below, we present the HCA system architecture, focusing on general architecture and information flow, and NISLab's natural language understanding, character modelling, and response generation modules. The system works in real time on a state-of-the-art PC and will be tested with target users in early 2004.

2. GENERAL ARCHITECTURE

The system's event driven, modular, asynchronous architecture is shown in Figure 1. In addition to the modules explained in more detail below, the modules are: a speech recogniser from partner Scansoft [12] (not used in PT1); a gesture recogniser based on the free OCHRE neural networks Java software [8]; gesture interpretation developed by partner LIMSI [5]; RealSpeak speech synthesis from Scansoft [10], including time calculation for animation tags; and character animation and virtual world simulation from partner Liquid Media [6]. The modules communicate via a central message broker, publicly available from KTH [4]. The broker is a server that routes function calls, results and error codes between modules. The Transmission Control Protocol (TCP) is used for communication. The broker coordinates input and output events by time-stamping all messages from the modules as well as associating them to a certain dialogue turn. The behaviour of the broker is controlled by a set of message-passing rules, specifying how to react when receiving a message of a certain type from one of the modules.

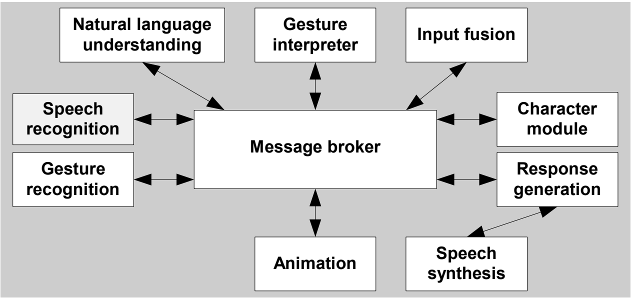

Figure 1. General NICE HCA system architecture.

Partner TeliaSonera [13] is developing natural language understanding, character modelling, and response generation modules for the fairy tale part of the system which otherwise draws on the components already mentioned. In the second prototype, the user will be able to go from HCA´s study to the fairytale world and interact with some of the characters from HCA's fairytales. In PT1, the two worlds remain separate.

In terms of information flow, the speech recogniser sends an n-best set of hypotheses (only in PT) to natural language understanding which sends a 1-best hypothesis to input fusion. Similarly, the gesture recogniser sends an n-best hypothesis set to the gesture interpreter which consults the animation module as to which object the user may have indicated. In PT1, the input fusion module simply forwards an n-best list of pairs of (recognised pointable object + gesture confidence score) from the gesture inter and/or a 1-best natural language understanding output to the character module which takes care of input fusion, when required. The character module sends a coordinated verbal/non-verbal output specification to the response generator which splits the output into synchronised text-to-speech and animation.

3. LANGUAGE UNDERSTANDING

NISLab's natural language understanding (NLU) module (Figure 2) is implemented in C++. It receives input from the speech recogniser (SR) and passes its output to input fusion (IF). The main recipient of NLU output is the character module (CM) via input fusion. The NLU output provides the CM with the necessary information to find an answer to the user input and plan HCA's next conversational move, cf. Section 4. The NLU has been designed to robustly handle the ungrammatical and only partially parsable user utterances characteristic of domain-oriented conversation. The components of the NLU are: a keyphrase spotter, a syntactic analyser, a domain/topic spotter and an FSA (Finite State Automaton) processor. The combination of keyphrase spotter and syntactic analyser acts as a shallow parsing component and the FSA processor as the deepest level of parsing. The NLU module manager manages internal NLU communication.

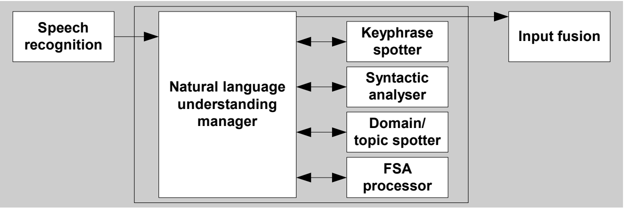

Figure 2. NICE HCA natural language understanding module.

The keyphrase spotter is the first NLU component. It has a set of keyphrases that map to syntactic/semantic categories. A semantic/syntactic category is an attribute-value pair. For instance, the semantic category <fairytale:ugly_duckling> has an attribute fairytale with ugly_duckling as its current value. An attribute can have multiple values. Thus, in the case of attribute fairytale, the values can be ugly_duckling, little_mermaid and princess_pea. The keyphrase spotter spots phrases in the user´s input and converts them into syntactic/semantic categories. A phrase example could be "I did not say..." which is mapped to the semantic category <meta:correction>. The output is passed on to the syntactic analyser.

The syntactic analyser consists of a number spotter, a lexicon, and a rule engine. The number spotter spots numbers in the input, indicating, e.g., the user's age. The lexicon entries consist of syntactic/semantic categories for individual words. Having passed the number spotter and the lexicon, the user input is a sequence of semantic and syntactic categories. The rule engine applies rules to this input sequence. The rules are defined based on the presence of certain semantic/syntactic categories at specific positions in the input sequence. The rules consist of a semantic/syntactic category sequence to be detected in the input sequence and a resulting semantic/syntactic category sequence that replaces it (see example below). The conversion depends on the presence or absence of conditions. A condition consists of the presence or absence of semantic/syntactic categories in a specific position.

The domain/topic spotter identifies the input topic(s) by mapping the semantic/syntactic categories to their respective topics. This mapping is defined at design time. Domains are identified based on topics. The result is sent to the FSA processor.

If the sequence is able to traverse an FSA, the result corresponding to the FSA is the NLU output semantics. An FSA compiler is used to develop the FSAs off-line from a training corpus. The utterances in the training corpus go through keyphrase and syntactic analysis to produce a syntactic/semantic category sequence. The sequence is used to form the FSAs. A sequence of semantic categories is defined for sets of FSAs at design time. The FSA processor reads these FSAs at system initialisation time. The values for the attributes in the resulting sequence are filled from the input user sequence. The result consisting of domain(s), topic(s) and semantics is sent to the input fusion module.

To illustrate NLU processing, consider the user utterance "I am ten years old". The keyphrase spotter does not change the utterance as it does not spot any keyphrases. The number spotter spots the number ten and the processed sequence is "I am <number:10> years old". The categories for individual words are taken from the lexicon, making the sequence: "<User><aux:am><number:10> <years><old>". The rule engine converts <User><aux:am> <number:10> to "<User_Info:age><number:10><years><old>". The Domain/Topic Spotter finds that only the semantic category <User_Info:age> has a mapping and is mapped to the topic "u_information". The domain "user" is identified from the topic. The FSA processor result is "<User_Info:age><number:10>". The resulting information on domain, topic and semantics is wrapped in XML and sent to the input fusion module.

4. CHARACTER MODULE

The HCA character module (Figure 3) is implemented in C++. It is managed by the character module manager which also takes care of module-external communication. The character module is in one of three output states, producing either: non-communicative action output, communicative function output, or communicative action. Non-communicative action (NCA) output is produced when nobody is talking to HCA. In this state, he is simply going about his work in his study. Communicative function (CF) output is produced when someone is talking and/or gesturing to HCA, to which he responds by showing friendly awareness of the user's input. For this to happen in real time, the character module has fast-track connections to the speech and gesture recognisers in order to act as soon as one of them receives input. Communicative action (CA) output is HCA's conversational contributions. The overall state relationships are: NCA → CF ↔ CA → NCA. Thus, NCA, the system's "resting state", must be followed by a CF state in which a new user starts addressing HCA. Following the user's first input, conversation in which user and system take turns may go on for a while, eventually being followed by the NCA state.

When a conversation with a user is ongoing, the character module manager (CMM) sends the input received from input fusion, i.e., a frame wrapped in XML, to the mind state agent manager (MSAM) of the mind state agent (MSA), cf. Figure 3. The MSA manages the user's spoken and/or gesture input, including the planning of which response to produce in reaction to the input.

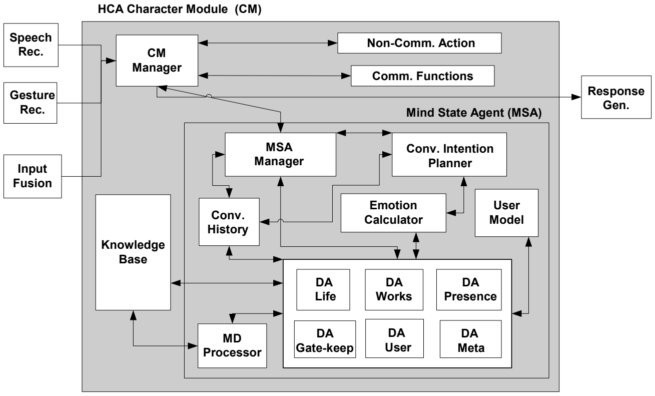

Figure 3. HCA character module architecture.

The MSAM is the central component in the mind state agent. Based on input and proposals from the conversation intention planner (CIP) which embodies HCA's conversational agenda, the MSAM communicates with the domain agents (DAs) to retrieve output. The default option is that HCA should, if possible, provide an in-domain response to the user's input. However, the CIP may decide that this should not be attempted because, e.g., the input was not understood and meta-communication should be initiated. The conversation intention planner will always propose a way in which to continue the dialogue, which may or may not relate to the topic and/or domain brought up by the user. The MSAM decides whether or not to use the proposed continuation. If the CIP has indicated that no attempt should be made to find a reply to the user's input, the MSAM will always use the continuation. Otherwise, the MSAM will try to retrieve a reply by contacting the DAs. If an empty reply is returned by the DAs, the continuation will always be used. If the retrieved reply was non-empty, the MSAM will, in most cases, randomly decide whether or not to use the continuation. A continuation may be a reference to a statement, a question, or a mini-dialogue (see below) which is retrieved from the DAs. The result will always be non-empty. This means that HCA will always say something, either in terms of a reply, a continuation, or both.

The MSAM may contact the DAs in three different ways. The two first ways mentioned below are used for retrieving a reply to user input and for retrieving a continuation, respectively, except when the domain is meta. In the meta case, Option 3 is used.

1. Function *get reply* takes as input a frame containing the user's input and HCA's emotional state. The return may be an output reference obtained from the knowledge base (KB) or null in case there is no reply in the KB for the user input.
2. Function *get continuation* takes as input a frame which, among other things, indicates the type of continuation, i.e. statement, question, or mini-dialogue. In case of a mini-dialogue, the function updates a variable in the conversation history (CH) about whether the mini-dialogue is starting, ongoing or ending. The return from the function will always be an output reference retrieved from the KB.
3. In case of meta, a function with a parameter action is used to indicate the particular meta case to be handled, e.g., *repeat.* The return will always be KB-retrieved output reference.

Internally, in the DAs component, the MSAM's request is directed to the relevant domain agent (Figure 3). For replies, the KB is always contacted directly by the DAs via an SQL query. For continuations, the KB is contacted directly in a similar way unless the proposed output is a mini-dialogue. A mini-dialogue is a predefined small dialogue that will allow HCA, on occasions at which he takes particular interest in the user's input, to carry out in-depth conversation on (a) certain topic(s). In case of a mini-dialogue, the DAs first contact the mini-dialogue processor (see below).

The user domain agent has the particular task of extracting from the frame received from the MSAM data related to a particular user, such as nationality, gender and age. This information is stored in the user model. The knowledge about age is used when selecting output in certain cases.

The mini-dialogue processor (MDP) processes mini-dialogues in a finite-state-machine approach. The MDP has two functions which may be called by the DAs. One is called only when a mini-dialogue starts. The second function is called at all other steps in an ongoing mini-dialogue. The latter function tells when the mini-dialogue ends. Both functions return an internal identifier retrieved from the KB. The identifier is used by the DAs to look up the relevant output references in the knowledge base.

The KB is implemented as an Access database which maintains the system's domains ontology, including references to all of HCA's coordinated spoken and non-verbal output.

Output references retrieved by the DAs from the KB are sent to the mind state agent manager which collects the returns from the DA function(s) called before sending the output for that turn to response generation via the character module manager.

A conversation history (CH), is maintained to keep track of information about the individual input and output turns in the conversation, such as the last domain and topic to which a response was looked up, the most recent domain and topic of a retrieved continuation, number of consecutive turns involving meta-communication, and mini-dialogue status (started, ongoing, ended). A second conversation record is maintained in a tree-structure by the conversation intention planner. This record keeps track of the domains and topics talked about so far, the topics and output already used, and the domains covered during conversation. HCA likes to talk to the user for a while before letting him/her into the fairy tale world. Thus, for each domain he wants a minimum number of exchanges before the domain is declared covered. Once a domain is covered, HCA will not return to it during the dialogue unless it is re-introduced by the user. The reason for keeping track of what has been said is to prevent HCA from repeating himself during conversation. Thus, replies to the user's input are repeated only if the user provides the same input several times.

The emotion calculator (EC) updates HCA's emotional state whenever the user's input produces an emotion increment which makes HCA more happy, sad, or angry. Emotion increments are attached to output references stored in the knowledge base. Whenever the domain agents query the KB, it is checked if there is an emotion increment. If there is, the EC is called and returns an updated emotional state. Since, in some cases, the exact phrasing of output depends on HCA's current emotional state, the KB is then queried again with the new emotional state.

5. RESPONSE GENERATION AND ANIMATION

Giving HCA a richer persona through emotion and personality modelling is not sufficient to create a life-like character [1,3]. To increase character believability, it also has to be able to display human-like behaviour by combining non-verbal (gesture, facial expression, pose, gaze) and verbal (speech) output during conversation. The response generator links emotional patterns and character animation in terms of speech as well as gesture [9].

The response generator must perform in real-time and must generate a comprehensive set of communicative and non-communicative actions, which can be rendered via speech synthesis and by the animation component. NISLab's response generator is implemented in C++ and Sicstus Prolog [11].

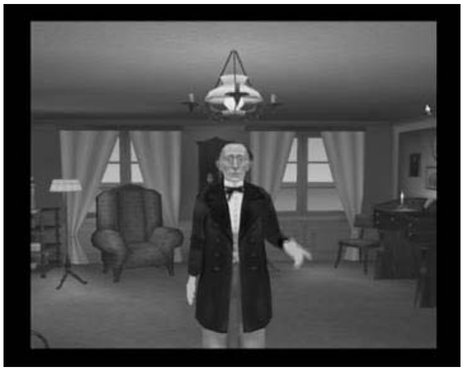

Figure 4. Uttering 'I would like to hear your opinion about the Prince' while performing a deictic gesture co-occurring with the word 'your' in the text template and after insertion of the word 'the Prince' in its variable value *FAIRY_TALE_CHAR.*

The response generator receives a parameterised semantic instruction composed of input values, text-to-speech (TTS) references, and/or references to non-verbal behaviours. The TTS references are used to retrieve text template output with embedded start and end tags for non-verbal behaviours (bookmarks) that are stored in the form '*I would like to hear [g0] your [/g0] opinion about {FAIRY_TALE_CHAR}*'. In this example, elements within brackets starting with numbered 'g' letters represent behavioural parts of the template. Elements within parentheses, like *FAIRY_TALE_-CHAR* in the example, represent variable values to be filled using contextual information delivered by the NLU during a conversational turn; 'the Prince' or 'the Little Mermaid' are two possible values associated to the fairy tales (representing contextual information) 'The Princess and the Pea' or 'The Little Mermaid', respectively. Both behavioural elements and variable values are initially uninstantiated and need to be retrieved at run-time. This approach allows for a high degree of flexibility as the binding of non-verbal behaviour to speech occurs at run-time rather than being hard-coded, enabling a sentence to be synthesised at different times with different accompanying gestures.

In order to provide timing information for speech and gesture during rendering, behavioural elements are made up of two sets of tags to indicate their start and end, respectively. Thus, in the previous example, tags *[g0]* and *[/g0]* indicate that a single movement has to co-occur while uttering the spoken text '*your'* around which they are wrapped. A pointing gesture may be attached to the gestural behaviour *[g0]* while uttering the short text. Figure 4 shows a snapshot of the animation generated with the text template in the example.

Once non-verbal-behaviour tags have been processed and variable values inserted into the templates, a surface language string results. The string is sent to the speech synthesiser which synthesises the verbal output and, when it meets a bookmark, sends a message to the response generator. The response generator creates an XML representation of the non-verbal element and sends it to the animation engine which takes care of the graphics output. The first NICE HCA prototype uses approximately 300 spoken utterance types and 100 different non-verbal behaviour primitives.

To animate the three-dimensional HCA character, we change its position, scale and orientation at different points in time. The animation system uses three scalar values to represent position and scale, and quaternion values to represent orientation. The character is built upon a hierarchy of elementary parts, referred to as frames, where each single frame represents a bone in the character. The hierarchy of frames together with a textured polygon mesh and skin-weight information is represented as a mesh with skin. The skin information specifies the influence a frame has on its mesh. The root frame contains a transformation matrix relative to the world space. An animation that affects the root node affects the whole scene while an animation that affects a leaf node does not affect any other node. Hence, the frame hierarchy gives the system the ability of playing single animations in parallel for different parts of the body to obtain complex animations.

6. ACKNOWLEDGMENTS

The work presented is being supported by the EU Human Language Technologies programme under contract IST-2001-35293. We gratefully acknowledge the support.

7. REFERENCES

[1] Argyle M.: Bodily Communication. 2nd edition, London and New York: Methuen & Co., 1986.

[2] Bernsen, N.O., Dybkjær, H. and Dybkjær, L.: Designing Interactive Speech Systems. From First Ideas to User Testing. London: Springer Verlag, 1998.

[3] Knapp, M.L.: Non-verbal Communication in Human Interaction. 2nd edition, New York: Holt, Rinehart and Winston Inc., 1978.

[4] http://www.speech.kth.se/broker

[5] http://www.limsi.fr/indexgb.html

[6] http://www.liquid.se/

[7] http://www.niceproject.com/

[8] http://sund.de/netze/applets/BPN/bpn2/ochre.html

[9] Picard, R.: Affective Computing. Cambridge, Mass.: MIT Press, 1997.

[10] http://www.scansoft.com/realspeak/

[11] http://www.sics.se/sicstus/

[12] http://www.scansoft.com

[13] http://www.teliasonera.se/