NICE project (IST-2001-35293)

Natural Interactive Communication for Edutainment

NICE Deliverable D5.2a

Second Prototype Version of Conversation Management and Response Planning for H.C. Andersen

29 October 2004

Editor

Niels Ole Bernsen

Authors

Niels Ole Bernsen, Marcela Charfuelán, Andrea Corradini, and Manish Mehta

NISLab, Odense, Denmark

Project ref. no.	IST-2001-35293
Project acronym	NICE
Deliverable status	Restricted
Contractual date of delivery	1 November 2004
Actual date of delivery	29 October 2004
Deliverable number	D5.2a
Deliverable title	Second Prototype Version of Conversation Management and Response Planning for H.C. Andersen
Nature	Report
Status & version	Pre-final
Number of pages	23
WP contributing to the deliverable	WP5
WP / Task responsible	TeliaSonera
Editor	Niels Ole Bernsen
Author(s)	Niels Ole Bernsen, Marcela Charfuelán, Andrea Corradini, and Manish Mehta
EC Project Officer	Mats Ljungqvist
Keywords	Conversation management, response planning, life-like animated characters
Abstract (for dissemination)	This report, Deliverable 5.2a from the HLT project Natural Interactive Communication for Edutainment (NICE), describes conversation management and response planning for life-like animated conversational agent H. C. Andersen in the second NICE prototype.

Table of Contents

1 Introduction

This NICE Report D5.1-2a describes conversation management and response planning for Hans Christian Andersen in the second NICE prototype (PT2). PT2 includes two generic life-like animated character software modules, one for Andersen (HCA) and one for the fairy tale characters. This is due to the fact that HCA and the fairy tale characters play very different roles in the NICE system. Whereas HCA is witty and conversational, but also rather static in a physical sense since he is confined to his study during interaction with users, the fairy-tale characters are physically active agents whose primary topic of conversation is limited to the gaming task and conventionalised social interchanges. The fairy tale character software module is described in D5.2b.

The WP5 description underlying this report emphasises the objective of character software re-use, stating that a software kernel will be developed which is the same for each character and which can accommodate implementation of different character profiles. Even though we will only be developing as single conversational character of the HCA type in NICE, i.e. HCA himself, it remains an important goal to develop the HCA character in such a way that the kernel character software can be re-used for other conversational characters of the same generic type as HCA. However, it must be kept in mind that we are dealing with a highly experimental software system, due to the fact that the HCA system is still the first of its kind, i.e. the first entire system which aims to demonstrate "real", i.e., domain-oriented rather than task-oriented, conversational interaction. For this reason, the transition from PT1 to PT2 has required us to address three different development objectives at the same time, i.e.:

- iterate the basic architecture, modularity, processing strategies, and information flow in our parts of the system in order to improve basic conversational abilities;
- evolve our parts of the system to accommodate basic system improvements provided by our partners, including (i) major improvements in the systems graphical rendering capabilities and hence in HCA's non-verbal behaviours, (ii) major modifications of the system's gesture input processing capabilities, (iii) inclusion of speech recognition, and (iv) replacement of the PT1 speech synthesis with improved TTS;
- develop for modularity and character re-use, both in the sense of (i) taking steps towards optimising the replacement of HCA with, for instance, Isaac Newton, and (ii) optimising the replacement of the current English language of conversation with, e.g., Danish or French.

These three objectives are not necessarily conflicting, of course, but their joint demands clearly makes the task of developing for PT2 re-use far more complex than it would have been had PT1 already embodied the many PT2 solutions which are different from those of PT1 and which, in fact, thoroughly affect most PT2 system components.

According to the WP5 description, the HCA kernel will implement the following computational steps:

1. resolution of discourse references, ellipses and deictic references when these have not been resolved by the input fusion module but require access to the dialogue history to be resolved;
2. dialogue act classification of user utterances for use in querying the characters' personalities and knowledge bases;
3. dialogue history for keeping track of the discourse context;
4. decision on the next communicative action(s) to be performed by the focal character, including meta-communication;

5. response planning.

In PT1, we had implemented Steps 3, 4 and 5. Contrary to what was stated in the predecessor to the present report, i.e. Report D5.1a, PT1 did not include implementation of the part of Step 1 which deals with deictic reference resolution relating to input fusion. PT1 did not include semantic input fusion at all. This part of Step 1 is being taken for PT2 at the time of writing. It is not clear at this point if PT2 will include first solutions to all of the remainder of Step 1, in particular, the use of dialogue history to resolve current input anaphora. PT2 implements Step 2. It should also be pointed out here that the, now rather old, WP5 description has been supplemented with the PT2 HCA system requirements specification presented in Report D1.1-2a "Requirements and design specification for domain information, personality information and dialogue behaviour for the second NICE HCA prototype".

The present report draws upon information provided in NICE reports D1.1-2a (just mentioned above) and D1.2-2a "Analysis and representation of domain information, personality information and conversation behaviour for H.C. Andersen in the second prototype". Report D1.2-2a describes in detail the extent to which the PT2 character module meets the PT2 requirements specification in D1.1-2a. In addition, D1.2-2a provides PT2 specifics on the representation of domain information and personality information for HCA, and describes the underlying design ideas behind the representation and coding of HCA's conversation behaviour in PT2. This information is not repeated in the present report. Thus, major chunks of PT2 HCA conversation management and response planning have been described in D1.2-2a already. In the present report, we put it all together, present the information flow model adopted for HCA in PT2, and describe, in the process, modules and functionalities which have not been described in D1.2-2a.

Another very important source of information for the present report are the WP2 data collection results from the January 2004 user test of HCA PT1. The results are described in Report D2.2a "NISLab's Collection and Analysis of Multimodal Speech and Gesture Data in an Edutainment Application". In addition, Report D1.2-2a summarises the weaknesses of PT1's conversation behaviour which we found by asking the users and by analysing the user test data.

The present report has been written too early to be able to take into account the contents of reports D3.2a on the English recogniser used in the HCA PT2 system, D3.5-2a on the HCA PT2 system's natural language understanding module, D3.6-2 on the multimodal input understanding module for HCA PT2, D3.7-2 on the multimodal output generation module for HCA PT2, and D4.2-2 on PT2 HCA rendering. The reader is referred to these reports for details on the system components in question. At the time of writing and, at least, partly according to PT2 development planning, we are still waiting for a second version of the English speech recogniser trained with the January 2004 user test data, the final version of PT2 rendering, and the PT2 gesture processing modules.

In what follows, Section 2 briefly describes what we consider to be the main challenge we are facing during development of conversation management and response planning for HCA in NICE PT2, i.e. the spoken conversation challenge. Section 3 presents the PT2 HCA character module architecture at a high level of detail and discusses differences in information flow compared to HCA PT1. Section 4 presents details of the PT2 HCA Mind state agent architecture and illustrates the corresponding character module information flow. Section 5 addresses the issue of software re-use.

2 The main challenge in PT2

As argued in several recent publications, e.g., [Bernsen and Dybkjær 2004a, 2004b, 2004c], a domain-oriented embodied conversation agent system, such as the NICE HCA system, requires at least two different theories at very different levels for its scientifically sound development. The *first theory* is a high-level theory of the general *type* of conversation between the user and the animated character. The HCA system's conversational behaviour is based on such a theory, i.e., the theory of successful prototypical human-human conversation described in the predecessor to the present report (D5.1-1a) which also discusses the notion of domain-oriented conversation. The *second theory* is actually required by the first theory and is a theory of conversational coherence for the agent's spoken domain-oriented conversation. This second theory is currently under development. In addition to providing explicit scientific foundations for system development, the theories are essential for theory-based evaluation of the system's conversational performance. To this end, we have developed several quantitative evaluation metrics based on the first, high-level, theory and applied those to the analysis of the January 2004 PT1 user test data [Bernsen 2004, Bernsen et al. 2004]. For completeness, it may be argued that a domain-oriented embodied conversation agent system, such as the NICE HCA system, also needs to be based on a *third theory,* i.e., a theory of the agent's non-verbal conversation. However, such a theory is far beyond the state of the art at the moment.

The January 2004 user test convincingly demonstrated, we submit, the validity of the high-level theory of conversation as instantiated by HCA's PT1 conversation, cf. [Bernsen and Dybkjær 2004c]. For us, this was important progress because it meant that we could proceed with HCA PT2 development without having to do basic high-level theory revision. However, the January 2004 user test also demonstrated that the *coherence* of HCA's conversation was far from perfect, thus emphasising the need for a theory of agent conversational coherence as well as for improving the coherence of HCA's conversation in PT2. We actually expected this result from the PT1 user test, of course, given the scale of the challenge of demonstrating human-style domain-oriented conversation. What the user test contributed, were a number of specific results on the coherence weaknesses in HCA's conversation. These weaknesses are presented in Report D1.2-2a and include:

- inflexibility of the PT1 mini-dialogue representation and processing approach which too often generates irrelevant output;
- too loose control of conversational continuations, generating irrelevant output;
- lack of appropriate response to generic input, including meta-communication;
- superfluous HCA repeat meta-communication requests when he should have been able to understand the user; and
- HCA's lack of ability to factor in the logical implications of user input in terms of which conversational topics to avoid.

Some of the weaknesses in the above list represent quite fundamental flaws in conversational coherence. If such weaknesses occur more than relatively few times in conversation, the conversation is perceived to fall apart into disjoint monologues.

Our development of PT2 conversation management and response planning for HCA thus has the goals of removing the weaknesses above and, in addition, meet the PT2 Character module requirements listed in Report D1.1-2a and discussed in Report D1.2-2a.

3 Architecture and high-level information flow

In this chapter, Figure 3.1 shows the general HCA PT2 architecture. Figure 3.2 shows the HCA PT2 Character module architecture.

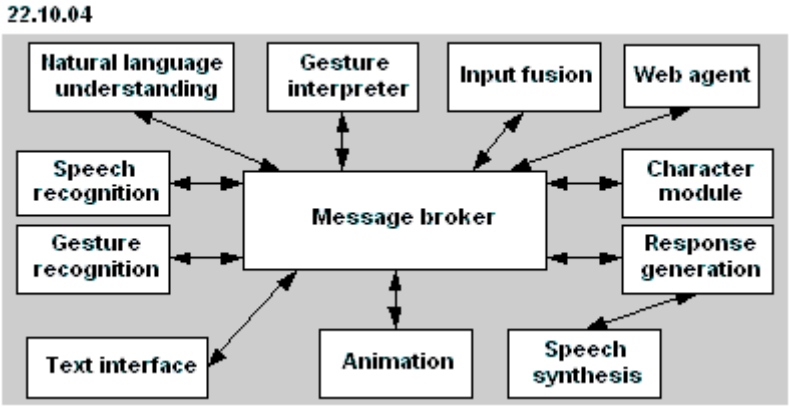

Figure 3.1. Overall NICE HCA PT2 system architecture.

3.1 General HCA PT2 architecture and information flow

Two of the three new modules in the overall HCA PT2 system architecture in Figure 3.1, i.e., the Text interface and the Web agent, are described in Report D1.2-2a. The speech recogniser is described in Report D3.2a. In terms of information flow, there are no major differences compared to PT1, cf. Report D5.1-1a.

3.2 HCA PT2 Character module architecture and information flow

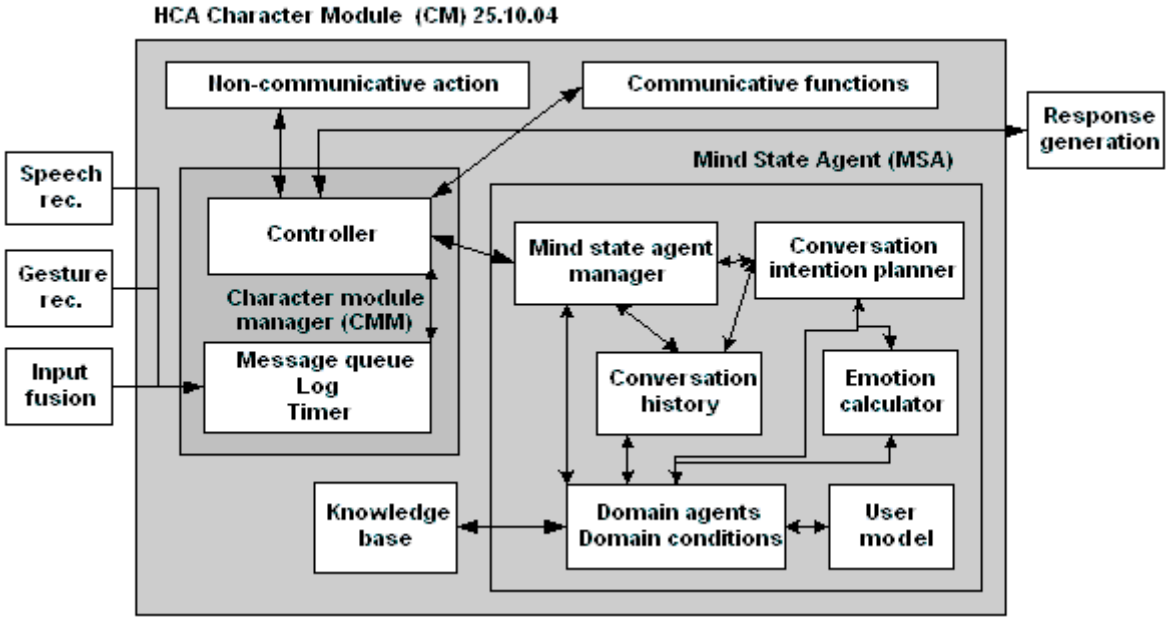

Figure 3.2. HCA Character module architecture for the second NICE HCA prototype.

Compared to HCA PT1, the PT2 Character module architecture is rather similar at the level of detail presented in Figure 3.2.

Due to its inflexibility, the FSM Mini-dialogue processor has been removed. PT2 still processes mini-dialogues but these are now embedded in the domain ontologies to which they belong, as explained in Report D1.2-2a.

As in PT1, the PT2 Character module is still always in one in three output states, i.e., Non-communicative action when HCA is alone in his study working; Communicative functions when HCA pays attention to the user's spoken input; and Communicative action when HCA actually responds to the user's input. As regards the processing of these output states, the main differences from PT1 are that (i) PT2 uses the more sophisticated non-verbal output generation functionalities provided by partner Liquid Media for PT2, and (ii) gesture input now also, like spoken input, has fast-track connections to Communicative functions, enabling HCA to pay immediate attention to the user's spoken and gesture input as soon as such input is detected by the speech recogniser and the gesture recogniser, respectively.

The more sophisticated PT2 non-verbal output generation functionalities referred to in the preceding paragraph include:

- basic speech/lip movement synchronisation;
- the possibility of rendering several non-verbal behaviours at the same time, so that, for instance, HCA can exhibit lip synchrony and gesture at the same time; and
- the possibility of specifying the start time and the duration/perceived speed and extent of execution of non-verbal action primitives, enabling more articulate and varied gesture, facial expression, body posture and movement behaviour, possibly concurrent with output speech;
- rendering engine information on HCA's position for path planning purposes..

Finally, the PT2 Character module knowledge base is completely new as regards architecture, role, and functionality, cf. Report D1.2-2a.

Compared to HCA PT1, the PT2 Character module information flow is rather similar at the level of detail presented in Figure 3.2, excepting, of course, the absence in PT2 of the Mini-dialogue processor. The Mind state agent manager controls the Character module, including the output state changes between Non-communicative action, Communicative functions, and Communicative action, cf. the state table, flow diagram, and behavioural element tables in Report D5.1a. The Mind state agent manager also controls the timing of various output pauses during which the system is listening for user input, and maintains an input stack. The Domain agents, User model, Emotion calculator, and Conversation history have the same basic roles and functionalities as in PT1. Modest extensions to the basic functionalities of the three first-mentioned components are described in Report D1.2-2a.

3.3 Natural interactive response planning

This section describes in more detail how PT2's augmented rendering functionality, described in Section 3.2, is being applied to PT2 non-verbal and combined verbal and non-verbal response planning. From a technical point of view, it may be useful to distinguish between two aspects of this question, i.e. response planning for non-communicative actions and communicative functions, on the one hand, and response planning for communicative actions on the other.

3.3.1 Non-communicative action and communicative function

Let us briefly remind the reader about what we refer to as non-communicative actions and communicative functions, respectively, from the perspective of the character module.

While in the communicative function output state, we have made our character show his awareness of being spoken to or otherwise addressed by the user, by employing a rather

‘neutral’ and general set of animations or sequences thereof. The relative ‘neutrality’ of these behaviours is imposed by the technical limitations which prevent us from processing the user’s input incrementally in real time. This means that he cannot react, *while* being addressed, to parts of the user input which, given his personality, should otherwise make him react emotionally or cognitively.

A non-communicative action output state refers to the situation in which HCA is not engaged in conversation with a user but goes about his normal work and life within his study. This may happen either at system startup if there is no user around, or after the user stops conversation and walks away. In general, while in this state, the character does not produce spoken utterances in terms of full-form sentences. Yet, the system is capable of playing back, e.g., footsteps when HCA walks, or music sounds when he, say, enacts a dance. Sound files in wav format and mp3 are stored to allow for that capability.

We believe all the two output states reported above are very important to accomplish the goal of eliciting a rich interaction experience.

In our implementation, we employ a hierarchical two-tiered approach that supports the designer in creating those output states through scripts. Lower-level scripts define sets of elementary animations along with their temporal specification and sets of custom animations. Elementary animations are behaviours that can be rendered by the animation engine by just one command and that belong to the core of the application as default animations. Custom animations are sequences of such elementary behaviours. Suppose, for example, that we have the following elementary animations (their names are self-explanatory):

OPEN_ARM
TURN_LEFT
TURN_RIGHT
THUMB_DOWN
RAISE_EYEBROWS
GOTO_PIC_COLOSSEUM
TURN_TO_USER
TURN_TO_WALL
POINTING
SMILE
KISS
TILT_HEAD

In this example, the following custom animations may be defined, e.g.:

CUSTOM_FLATTER_USER = [SMILE, KISS, TILT_HEAD]
CUSTOM_GO_POINT_COLOSSEUM = [GOTO_PIC_COLOSSEUM, POINTING, TURN_TO_USER]
CUSTOM_DON’T_KNOW = [OPEN_ARM, RAISE_EYBROWS, TURN_TO_WALL]

Thus, e.g., any time HCA is requested to perform a CUSTOM_DON’T_KNOW animation, this be broken down into its two constituent elementary animations OPEN_ARM and RAISE_EYBROWS that are played in sequence. Recursive calls within custom animations are allowed.

At a higher level, sequences of elementary and/or custom animations are used to define complex behaviours. To make the character perform a richer variety of behaviours, we make use of placeholders within those sequences for animations to be selected at run-time. In more detail, we define several sub-sets of behaviours, assign them a name, and select them in a non-deterministic way, one animation each time the placeholder points to that particular subset. We also use syntactic rules to define and provide appropriate transitions among animations in order to produce believable and smooth interactive character behaviour.

An example is the following. Given the following sub-set for defining a new set of animations called SUBSET_TURN that contains the two behaviours TURN_LEFT and TURN_RIGHT:

SUBSET_TURN = [TURN_LEFT, TURN_RIGHT]

the following transition rule:

1 CUSTOM_FLATTER_USER CANNOT FOLLOW CUSTOM_DON'T_KNOW

the scripts:

SC_1 = [CUSTOM_GO_POINT_COLOSSEUM WAIT 1, CUSTOM_FLATTER_USER WAIT 2]

make HCA perform the behaviours described by the animation CUSTOM_GO_POINT_COLOSSEUM at the lower hierarchical level, wait for 1 second and then perform the other lower-level animation CUSTOM_FLATTER_USER and eventually wait for 2 seconds.

Differently from SC_1 above, the script SC_2 defined as:

SC_2 = [CUSTOM_GO_POINT_COLOSSEUM WAIT 1, SUBSET_TURN WAIT 2]

- is more flexible and has more variability through the use of sub-sets for the appropriate animation to follow the behaviour CUSTOM_GO_POINT_COLOSSEUM which has to be chosen in a non deterministic way at random time.

We use syntactic rules to define and to provide appropriate transitions among scripts. So, for example, the rule:

SC_1 CANNOT FOLLOW SC_2

- states that script SC_1, if chosen at run time and after being realised, cannot be followed by script SC_2. In that case, the Character module, in which the decision about which script to run resides, is constrained in its choice of the next script yet it is guaranteed that script transitions occur smoothly without abrupt movements between the end of a script and the start of the following one.

3.3.2 Communicative action

When conversation is going on between the user and HCA and it is HCA's turn, the Character module looks for HCA's conversation contribution within a knowledge base that stores many predefined sentences along with encoded non-verbal behaviours, semantic classes, and the system's domain ontology. Numerous, ever-expanding in number, canned templates guarantee broad domain coverage but also require manual maintenance and have a limited variability by design. Each template is a compact representation of a predefined spoken output with embedded start and end tags for non-verbal behaviours and placeholders to text values to be filled in at run-time. The following is an example of behavioural template:

Now, tell me [g0] your [/g0] {EMOTION ADJ_2} opinion about {FAIRYTALE}

Here, elements within square brackets starting with numbered *g* letters represent onset and offset of non-specified non-verbal parts of the template. Elements within curly parentheses, like *FAIRYTALE*, are placeholders for text-to-speech values to be filled in using input value information. The other elements within curly parentheses, starting with the string *EMOTION*, are TTS values as well but these are tied to emotional values. In the present example, *ADJ_2* indicates a set of emotional value/text pairs from which the verbal realisation for the appropriate text has to be retrieved. Both TTS variable values and non-verbal behavioural elements are initially uninstantiated. The binding of non-verbal behaviour to gesture and TTS variables to text occurs at run-time rather than being hard-coded, enabling a sentence to be synthesised at different times with different accompanying non-verbal elements and/or words.

The pair of tags that marks start and end of any non-verbal behavioural element supplies implicit timing information for speech and gesture during rendering. In the behavioural template above, tags *[g0]* and *[/g0]* indicate that an animation may co-occur with uttering the spoken text '*your'* around which they are wrapped. A certain gesture is selected for insertion in place of *g0* depending on the semantic class(es) of the text surrounded by the placeholders. Tables that map semantic categories onto non-verbal behaviours are maintained. Let us assume that a *POINT* animation is selected to expand the non-verbal behaviour *g0*, while the textural placeholders *EMOTION ADJ_2* and *FAIRYTALE* are expanded to *valuable* and *the Princess and the Pea*, respectively. The behavioural template is then converted into the surface language string:

Now, tell me [POINT] your [/POINT] valuable opinion about the Princess and the Pea

We have been implementing a strategy different from the one in PT1 to deal with such surface representation. The RG still replaces non-verbal behaviour references with bookmarks that can be dealt with by a text-to-speech component. Then, the entire string containing the TTS bookmarks is sent to the TTS, which synthesises the verbal output. Any time a bookmark is encountered, the TTS fires an event and calls on the Response generator to create the XML string representation of the corresponding animation. In PT2 however, we first parse the surface string for the TTS module to create wav files of text enclosed within animation bookmarks and determine its temporal duration. During parsing, the surface string is broken down into sequential segments of either audio-only segments, animation-only segments, or parallel audio and video segments. Three XML strings would be generated when parsing the surface string of our previous example:

1) <play> <sound> SOUND_1 </sound> </play>

2)<play>
<sound> SOUND_2 </sound>
<animList Track=0> 0, POINT; </animList>
</play>
3) <play> <sound> SOUND_3 </sound> </play>

Here, *SOUND_1* contains the synthesized text *now tell me*, *SOUND_2* the text *your*, and eventually the verbal synthesis for *valuable opinion about the Princess and the Pea* is stored into *SOUND_3*. The animation *POINT* is stretched over a time period equivalent to the duration of the sound file *SOUND_2*. Once all XML segments are created, they are sequentially sent to the graphical animation engine that automatically coordinates playback of sound and non-verbal behaviour rendering for each of them. This approach is suitable for short behavioural templates because it requires the data to be analysed twice: first parsing the template to create single wav files, then go through it again to break it down into single segments to send to the animation engine. Long templates are a technical issue wrt. this approach. Thus, we prefer to break down as many of them as possible as sequences of shorter ones.

The approach is, nevertheless, technically more challenging than the one followed in PT1, but we wish to mention the problems that occur in fine-tuning the duration of single animations for each of them to last exactly as long as the sound files they play along with, independently of the machine that runs the application. These technical problems are particular visible in the rendering of synchronised visemes and speech to be played back.

In addition to elementary animations, more complex non-verbal behaviours can be created, combined, sequenced, assigned a name, and stored by the RG. To that end, we have employed a generative approach based on a layered composition of primitives similar to that described in the previous sub-section. This process consists of the following hierarchically arranged functional elements. First, complex behaviours are designed as sequences of primitives. At the next level, such sequences are assigned priorities and composed to run concurrently. Finally, at the top-most level, scripts, i.e., sequences of complex animations, are defined to synthesise long, continuous, non-repetitive behaviours and to allow for smooth transitions between them. Since transitions from certain behaviours into some others may sometimes result in awkward motions, we use rules to either allow or prohibit transitions. In addition, to have high variability in our scripts and thus non-deterministic synthesis of motions, we allow behaviours within a script to be chosen at run-time from subsets defined by the designer. For example, let us say we have a subset *TURN* made up of {*TURN_RIGHT*, *TURN_LEFT*} behaviours and a script *THINK_THRU* defined as sequence of [*SCRATCH_HEAD*; *WAIT 3, RANDOM FROM TURN; WAIT 2, FROWNED_EYES; WAIT 3*]. Any time the script *THINK_THRU* is executed, it sequentially makes HCA scratch his head, wait for 3 seconds, choose randomly an animation from subset *TURN*, wait for 2 seconds, frown, and eventually wait for 2 more seconds. Animations within subsets can also be weighted to bias the random selection. Sets of rules over the scripts/behaviours ensure smooth transitions between scripts/behaviours.

Since the rendering engine can play only elementary animations, the Response generator has to break any user-defined animation down into its primitive components and create XML representation strings for each of them. *THINK_THRU* would thus have to be split into its three defined components at run-time. In turn, these latter have to be recursively decomposed until only primitives are used to express the parent animation *THINK_THRU*. Sequentiality, parallelism, and partial overlapping of existing animations to create new behaviours can be tuned by setting appropriate values for the temporal items in the XML representation.

Currently, we store some 300 output templates, many of which are no-variable stories to be told by HCA, and 110 different non-verbal primitives. Templates have been designed by hand and, similarly to non-verbal behaviours, were partly inspired by analysis of data from recordings of an actor impersonating HCA and interacting with kids in a children's theatre class in the fairy-tale writer's hometown Odense, Denmark.

Detailed information flow, an example

3.4 HCA PT2 Mind state agent architecture details

Figure 4.1 shows details on the Mind state agent manager and the Conversation intention planner. When we look at the details of the PT2 Mind state agent in Figure 4.1, we find some major innovations compared to PT1.

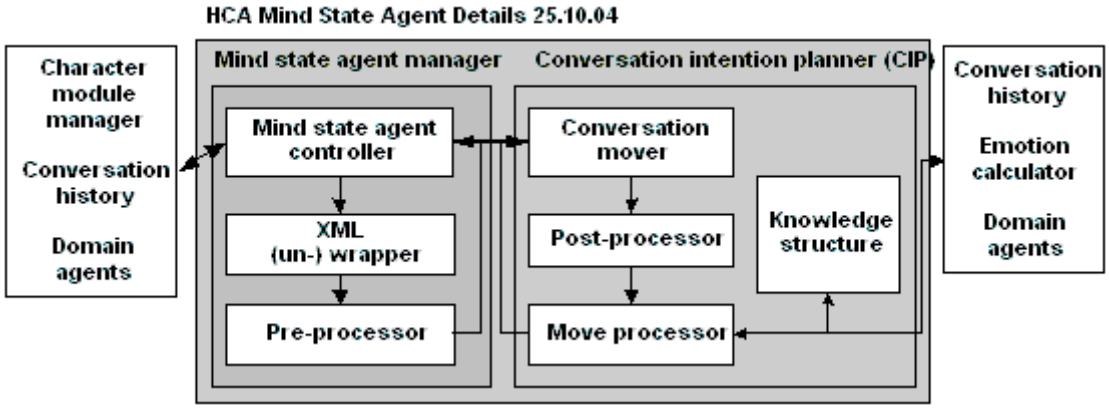

Figure 4.1. HCA PT2 Mind state agent details.

The Mind state agent manager XML (un-) wrapper unwraps the XML input frame and eventually wraps into XML each frame processed by the Mind state agent before the frame is sent to Response generation.

3.5 HCA PT2 Mind state agent information flow

In the following, we illustrate PT2 Mind state agent processing through an example. Suppose that the user says, in the context of conversation about HCA's life: "Do you like Odense?". The XML frame from the Natural language understanding module is:

```
<?xml version='1.0' encoding='ISO-8859-1'?><!DOCTYPE nluframe SYSTEM
'nlu_nice.dtd'>
<nluframe><date>2004-10-26 16:9:31 </date>
<speech_input>do you like odense</speech_input>
<semantic cs='2'>
   <number_of_domain val='1'>
       <domain val='life'>
            <number_of_concept val='1'>
                <concept val='location'>
                  <number_of_subconcept val='1'>
                      <subconcept val='odense'></subconcept>
                  </number_of_subconcept>
                </concept>
            </number_of_concept>
      </domain>
   </number_of_domain>
   <number_of_property val='1'>
      <property val='like'>
            <number_of_property_type val='1'>
```

```
                <property_type val='no_value'></property_type>
            </number_of_property_type>
        </property>
    </number_of_property>
    <number_of_dialogue_act val='1'>
        <dialogue_act val='question'>
            <number_of_dialogue_act_type val='1'>
                <dialogue_act_type val='yes/no'></dialogue_act_type>
            </number_of_dialogue_act_type>
        </dialogue_act>
    </number_of_dialogue_act>
    <number_of_dialogue_act_subject val='1'>
        <dialogue_act_subject val='hca'></dialogue_act_subject>
    </number_of_dialogue_act_subject>
</semantic>
</nluframe>
```

Figure 4.2. XML frame from Natural language understanding.

For a complete XML input frame from the Input fusion module, see Report D3.6-2. In the present example, the unwrapped input frame is:

```
Input_Frame:
NLU CS: 2
Number Of Domains: 1
Domain name: life
Number Of Concepts: 1
Concept: location
Number Of SubConcepts: 1
Subconcept: odense
Number Of Properties: 1
Property: like
Property Type: no_value
Number Of Dialogue Act Subjects: 1
Dialogue Act Subject: hca
Number Of Dialogue Acts: 1
Dialogue Act: question
Dialogue Act Type: yes/no
```

Figure 4.3. Unwrapped XML frame, cf. Figure 4.2.

Following processing by the Mind state agent manager Pre-processor, the frame is sent to the Conversation intention planner's Conversation mover.

The Conversation mover is a new module which has been built in order to process the input from the Natural language understanding module which has been significantly revised for PT2 purposes, cf. Report D3.5-2a, and from the Input fusion module which has also been significantly revised for PT2, cf. Report D3.6-2.

In PT2, the Natural language understanding module represents the user's spoken input in terms of semantic concepts which reflect both the current input contents and its dialogue act type. Using the input example above, i.e., "Do you like Odense?" (Odense being HCA's hometown), the Conversation mover is interested in the following parts of the frame, shown below in its stand-alone test version:

```
Concept Recognizer Output: <dialogue_act:question>
<dialogue_act_type:yes/no> <dialogue_act_subject:hca>
<property:like> <concept:location> <sub_concept:odense>
```

Figure 4.4. What the Conversation mover looks at, cf. Figure 4.3.

In the example, the Natural language understanding module represents the user's input as a *question* dialogue act of type *yes/no.* The *subject* of the dialogue act is *HCA. Properties* may have particular values but in the present case the input only involves the generic property *like.* Similar to properties, input *concepts* can have *sub-concepts* which specialise the generic concept. In the present example, the input involves a concept/sub-concept pair, i.e., *location: Odense.*

The Conversation mover's task is to attempt to match the ontological representation of the user's input to available HCA output. This is done through *key semantics* rules which range over the dialogue acts, concepts and properties in the ontological representation received by the Conversation mover. The user input only matches one or several system outputs if the rules for that (those) output(s) achieve a match with the ontological representation. In the present case, the input ontological representation matches the output key semantics defined on the following ontological categories:

```
dialogue act type: no_value or general or yes/no or listen or
positive or location
concept: location, sub-concept: birthplace or odense or hometown
```

Figure 4.5. Output key semantics example.

In addition, the input ontology attribute/values dialogue act: *question,* subject: *HCA,* and property: *like* satisfy the rule for the identified output. Furthermore, the input ontological representation does not match any other rule. We call this a *perfect match* which the Conversation mover outputs as:

```
Conversation Move:
Cmover domain[0]: life
Conv_move[0]: hometown_story
```

Figure 4.6. Conversation mover output.

In the present example, the Conversation mover Post-processor can simply forward the output Shown in Figure 4.6 to the Conversation intention planner's Move processor (Figure 4.1).

The Move processor is a new module which has been built in order to process the input from the Conversation mover and its Post-processor. The functionality of the Move processor is described in Report D1.2-2a, as are the functionalities of the User model, Emotion Calculator, Knowledge structure, and Domain agents, with which the Move processor communicates.

In the input example above, the input is not, e.g., Meta domain or Gatekeeper input and hence does not involve meta-communication or Gatekeeper communication, action by the Meta Domain agent or the Gatekeeper Domain agent, consultation with the Conversation history on preceding Meta-communication or Gatekeeper communication (patterns in meta-communication or Gatekeeper communication), and possible update of HCA's emotional state as a result of the accumulating history of meta-communication or Gatekeeper communication (Report D5.1-1a). Furthermore, the input does not cause any other emotional increments (Report D1.2-2a). Had this been the case, the Move processor would need to update HCA-s emotional state with the input-generated increment. Nor is the User model involved (Report D1.2-2a). Had this been the case, the Move processor would need the User Domain agent to

update the User model. Finally, the Conversation history does not cause modifications to the output. In other words, the Move processor is free to simply retrieve from the Knowledge base, via the Domain agents, the id for output identified by the Conversation mover, i.e.:

> *<g0> When I was born and lived in Odense as a child </g0> it was the second largest city of Denmark <g1> </g1> after the capital Copenhagen. Around 5000 people lived there, <g2> and the crown prince himself </g2> <g3> </g3> governed the city. I liked the city very much. The streets looked nearly as they had in the Middle Ages, <g4> </g4> most of the streets had <g5> not even cobblestones </g5> but were muddy and dirty. Many customs <g6> </g6> were observed here which had disappeared long time before in Copenhagen. For instance you could sometimes see women <g7> </g7> carrying the yoke, a wooden punishment instrument, <g8> </g8> around the neck. This was a punishment <g9> </g9> women got at that time for spreading gossip and lies.*
>
> *g0 = OPEN_ARMS animation*
>
> *g1 = NOD animation*
>
> *g2 = EXTEND_LEFT_ARM animation*
>
> *g3 = NOD animation*
>
> *g4 =SHRUG animation*
>
> *g5 =SHAKE_HEAD animation*
>
> *g6 =NOD animation*
>
> *g7 =RAISE_EYEBROWS animation*
>
> *g8 =SHRUG animation*
>
> *g9 =WRINKLE animation*

Following the decision to generate the spoken output story above, including its associated non-verbal behaviours, the Move processor algorithm requires a pause to be generated in the output following the story. This pause will allow the user to comment on the story told by HCA, ask follow-up questions, do meta-communication, do other generic communication (Report D1.2-2a), change topic or domain, or say nothing at all. In addition, the Move processor checks the Conversation intention planner's conversation agenda (Report D1.2-2a) in order to check if any agenda priorities may interfere with the default continuation of conversation about HCA's life. Let us assume that this is not the case. Had it been the case, the Move processor would have produced continuation output which changed the domain to a conversation domain which currently holds higher priority than the Life domain.

Following the pause, and provided that the user has not wished to change domain and that the current Life domain sub-segment on HCA's childhood currently addressed still includes stories to tell which have not been told before, the Move processor decides to offer the user another story in the current Life domain sub-segment and dynamically selects the story to offer. Let us suppose that the next unblocked story in the current segment of the Life domain in the Conversation intention planner's Knowledge structure is a story about how young HCA and his family lived in Odense.

In this case, the Move processor retrieves the story id from the Knowledge base. The Move processor also requests the Emotion calculator for an update of HCA's emotional state, including an incremental decay.

In addition, the Move processor updates the Conversation history, blocks the story told in the Knowledge structure, and considers the predictions for the next user input retrieved from the knowledge base (Report D1.2-2a).

The id of the output story to be told and the conversation continuation id identified, together with their non-verbal counterparts and HCA's current emotional state, and including also the type of continuation identified, are then sent to the Response generator.

The Response generator processes this input and creates the surface language needed in order to offer the continuation. In the present example, this requires the Response generator to combine surface language corresponding to the type of continuation, i.e., the Life sub-segment continuation expressed as, e.g., "You want to hear about", with surface language corresponding to the selected output continuation label, i.e., the childhood_home_story expressed as "how we lived", getting, e.g.:

<g0> <g1> You want </g1> </g0> to hear <g2> </g2> about how we lived?

g0 = RAISE_EYBROWS animation

g1 = FURROW_BETWEEN_EYES animation

g2 = TILT_HEAD animation

Following that, the Response generator times the verbal and non-verbal output, and sends the generated verbal and non-verbal output for realisation through TTS and graphics rendering (see Report D3.7-2).

3.6 Conversation history

Like in PT1, the PT2 HCA Character module includes a Conversation history which keeps track of the context of the conversation in terms of what has happened in previous user-HCA exchanges. The conversation history keeps a record of the frame resulting from each exchange, i.e., a frame which includes the user's input, references to HCA's output, and the intermediate values produced by the mind state agent module. In addition, the Conversation history keeps track of communication patterns, such as repeated occurrences of meta-communication.

Unlike PT1, however, the PT2 HCA Character module includes a Knowledge structure (Figure 4.1) which shares a significant part of the load of recording the conversation history. The Knowledge structure does not keep a turn-by-turn record of the conversation. Rather, the Knowledge structure is a linked set of domain tree structures which represents the domain ontological structures, keeps track of blocked segments, i.e., segments which have already been used for output in the current conversation, and includes intra-domain relevance links and mini-dialogue structures which help determine how to continue the conversation at any given point.

4 System re-usability

We no return to the issue of system re-usability introduced in Chapter 1. This issue represents one of the state-of-the-art research challenges involved in developing the HCA Character module, i.e., the 'kernel' challenge of building software which is easily re-usable for realising characters with knowledge and personalities very different from those of HCA. Both the WP5 description in the contract and the HCA PT2 specification underlying this report mention the goal of being able to parameterise the character kernel by a body of domain knowledge and a personality trait specification described in a representation language to be developed.

In Chapter 1 above, we have pointed out the difficulty of uniquely focusing on character re-use in a situation of software development in which all components are undergoing continual modification. Still, we need to address the question concerning the extent to which the PT2 Character module software system can be easily re-used for building other characters.

By way of example, let us suppose that we want to replace HCA with Sir Isaac Newton and ask which major system modifications this would require. Let us assume, in addition, that no revisions will be required of the system and component architectures shown in Figures 3.1, 3.2, and 4.1. In other words, since the system architecture is not assumed to have to undergo any significant change, what we are asking about are the modifications which need to be made within the existing architectural framework. This question may be somewhat less simple than might appear at first glance. So, let us first stipulate a number of high-level requirements:

- the user group is the almost the same as for HCA, i.e., the 14-18 years old who might take an interest in meeting one of the giants in the history of the natural sciences;
- the system still has an edutainment purpose, i.e., the combined purpose of historical correctness of the information conveyed and of entertaining conversation between Newton and young people interested in the origins of today's natural sciences;
- basically, our high-level theory governing HCA's conversation will also apply to Newton, cf. Report D1.2-2a;
- the use setting for Newton will be similar to the one for HCA, i.e., a museum or similar setting in which we envision 5-20 minutes of conversation with each user;
- the Newton system will be a research prototype just like the HCA PT2 system, i.e., one in which the goal of providing conversation which is as natural as possible overrides the goal of providing as much information as possible;
- of course, the Newton system must be of a quality similar to that of the HCA system as measured in terms of the NICE arsenal of quantitative, qualitative, and subjective parameters, cf. Report D7.1.

Given the above requirements, what will it take to replace HCA by Newton? It seems that we will need to do the following, at least:

1. select for Newton, say, the domains of Meta, Life, Person, Study, and Works. This means that we can re-use the PT2 HCA design libraries for, at least, Meta, Life, Person and Study;
2. as regards the Works domain, we will focus on one major aspect of Newton's work, such as his work on mechanics, just like we focus on HCA's fairytales;
3. given the difference between fairytales and mechanics, we may have to create a new design library volume (a domain ontology, including possible mini-dialogues) on conversation on mechanics. However, we would regard this as a matter for further analysis rather than conceding the full point straightaway. Clearly, some new mini-

dialogues will be needed but these can be unceremoniously plugged into the domain ontologies;

4. simulate the Newton design specification once to collect data for recogniser training, in particular language modelling, for Natural language understanding modification, Conversation mover modification, and Newton domain ontology tuning;
5. make the resulting modifications to recogniser language modelling, natural language understanding, the Conversation mover, and the domain ontologies;
6. design new output based on Newton contents acquisition and the ensuing global grasp of who he is and what he knows;
7. morph graphical HCA into Newton and HCA's study into Newton's study;
8. re-use the HCA English speech synthesiser;
9. any other issue arising, such as the opportunity to improve the PT2 HCA system kernel in particular respects;
10. system and component testing, including user testing; and
11. final modifications.

These would seem to be the principal points. Among these points, Points 1 and 2 have been done above already. Point 3 *may* require design following the global design pattern used for the PT2 HCA system's domain. This will not represent a major effort. On average, at least one effort of this type may be assumed to be needed during the development of the first 5-10 new characters within the global domain-oriented conversation system type outlined above. Point 4 requires substantial effort but we are thoroughly familiar with this kind of work. Point 5 includes routine language modelling work, an at this point unknown amount of work on natural language understanding, Conversation mover, and domain ontology modification. Point 6, the knowledge acquisition for a new character will remain a substantial task, whatever the character chosen, as will be the verbal and non-verbal output design. Point 7 is as easy as can be when replacing HCA with Newton. However, other replacements may require more substantial graphics rendering work. Likewise, Point 8 is no problem at all when replacing HCA by Newton. However, different synthesisers may be required when embodying female or young characters. Finally, Point 9 is a must to be kept in mind in any early-stage character replacement process, simply because modifications may serve to optimise subsequent replacements. Points 10 and 11 are obvious.

5 References

5.1 Publications

Bernsen, N. O.: Measuring relative target user group success in spoken conversation for edutainment. In J.-C. Martin et al. (Eds.): *Proceedings of the LREC 2004 Satellite Workshop on Multimodal Corpora: Models of Human Behaviour for the Specification and Evaluation of Multimodal Input and Output Interfaces.* Lisbon, Portugal, 25.5.04. Paris: European Language Resources Association (ELRA), 17-20.

Bernsen, N. O. and Dybkjær, L.: Domain-oriented conversation with H.C. Andersen. In E. André, L. Dybkjær, W. Minker, and P. Heisterkamp (Eds.): *Proceedings of the Tutorial and Research Workshop on Affective Dialogue Systems (ADS04)*, Kloster Irsee, Germany, June 2004a. Heidelberg: Springer Verlag: Lecture Notes in Artificial Intelligence, Vol. 3068, 142-153.

Bernsen, N. O. and Dybkjær, L.: Structured interview-based evaluation of spoken multimodal conversation with H.C. Andersen. *Proceedings of The International Conference for Spoken Language Processing, ICSLP 2004*, South Korea, October 2004b. International Speech Communication Association (ISCA) 2004, Vol. 1, 277-280.

Bernsen, N. O. and Dybkjær, L.: Evaluation of Spoken Multimodal Conversation. *Proceedings of The Sixth International Conference on Multimodal Interaction, ICMI 2004*, Pennsylvania, USA, October 2004a. New York: Association for Computing Machinery (ACM) 2004, 38-45.

Bernsen, N. O., Dybkjær, L., and Kiilerich, S.: Evaluating conversation with Hans Christian Andersen. In M. T. Lino et al. (Eds.): *Proceedings of the Fourth International Conference on Language Resources and Evaluation* (LREC'04), Lisbon, Portugal, May 25-28, 2004. Paris: European Language Resources Association (ELRA), Vol. 3, 1011-1014.

5.2 NICE reports

D1.1-2a: Bernsen, N. O. et al.: Requirements and design specification for domain information, personality information and dialogue behaviour for the second NICE HCA prototype. HLT project NICE Report D1.1-2a. NISLab, Denmark, April 2004.

D1.2-2a: Bernsen, N. O., Marcela Charfuelán, Laila Dybkjær, Abhishek Kaushik, Mykola Kolodnytsky and Manish Mehta: Analysis and representation of domain information, personality information and conversation behaviour for H.C. Andersen in the second prototype. HLT project NICE Report D1.2-2a. NISLab, Denmark, October 2004.

D2.2a: Bernsen, N. O., Laila Dybkjær, and Svend Kiilerich: NISLab's Collection and Analysis of Multimodal Speech and Gesture Data in an Edutainment Application. HLT project NICE Report D2.2a. NISLab, Denmark, April 2004.

D3.2a: Scansoft and NISLab: English recogniser for the HCA PT2 system. HLT project NICE Report D3.2a. Scansoft, Germany, November 2004 (to appear).

D3.5-2a: Mehta, M. et al.: English natural language understanding for the HCA PT2 system. HLT project NICE Report D3.5-2a. NISLab, Denmark, November 2004 (to appear).

D3.6-2: LIMSI et al.: Multimodal input understanding module for the second prototype. HLT project NICE Report D3.6-2. LIMSI, France, November 2004 (to appear).

D3.7-2: LIMSI et al.: Multimodal output generation module for the second prototype. HLT project NICE Report D3.7-2. LIMSI, France, November 2004 (to appear).

D4.2-2: Liquid Media: Second version of the system prototype, inhabited by characters from H. C. Andersen's world. HLT project NICE Report D4.2-2. Liquid Media, Sweden, October 2004 (to appear).

D5.1a: Bernsen, N. O. and Laila Dybkjær: First prototype version of dialogue management and response planning. HLT project NICE Report D5.1a. NISLab, Denmark, October 2003.

D5.2b: Telia: Second prototype version of dialogue management and response planning for the fairytale characters. HLT project NICE Report D5.2b. Telia, Sweden, October 2004 (to appear).

D7.1: Dybkjær, L., Niels Ole Bernsen, Reinhard Blasig, Stéphanie Buisine, Morgan Fredriksson, Joakim Gustafson, Jean-Claude Martin, Mats Wirén: Evaluation criteria and evaluation plan. HLT project NICE Report D7.1. NISLab, Denmark, March 2003.