NICE project (IST-2001-35293)

Natural Interactive Communication for Edutainment

NICE Deliverable D1.1

Requirements and design specification for domain information, personality information and dialogue behaviour for the first NICE prototype

30 September 2002

Authors

Niels Ole Bernsen[1], Johan Boye[2], Svend Kiilerich[1] and Ulrik Lindahl[3]

1: NISLab, Odense, Denmark, 2: Telia Research AB, Farsta, Sweden, 3: Liquid Media, Stockholm, Sweden

Project ref. no.	IST-2001-35293
Project acronym	NICE
Deliverable status	Internal
Contractual date of delivery	31 August 2002
Actual date of delivery	30 September 2002
Deliverable number	D1.1
Deliverable title	Requirements and design specification for domain information, personality information and dialogue behaviour for the first prototype
Nature	Report
Status & version	Final
Number of pages	30
WP contributing to the deliverable	WP1
WP / Task responsible	Niels Ole Bernsen
Editor	Niels Ole Bernsen
Author(s)	Niels Ole Bernsen, Johan Boye, Svend Kiilerich and Ulrik Lindahl
EC Project Officer	Mats Ljungqvist
Keywords	Edutainment, natural interactivity, conversational spoken dialogue systems, speech in computer games, multimodal interaction, human language technologies.
Abstract (for dissemination)	This deliverable from the IST/HLT NICE (Natural Interactive Communication for Edutainment) project specifies the domain information to be included in the first system prototype as well as the personality and dialogue behaviour specific to the embodied characters to be developed.

Table of Contents

List of authors and contributions

NICE Deliverable D1.1 was edited by Niels Ole Bernsen.

Chapters 1 and 2 were written by Niels Ole Bernsen with contributions from Svend Kiilerich who did the topics lists and Laila Dybkjær who made the Visio diagram in Figure 1.

Chapter 3 was written by Johan Boye and Ulrik Lindahl.

Requirements and design specification for domain information, personality information and dialogue behaviour for the first NICE prototype

1. Introduction

1.1 Goal

This deliverable describes our current requirements and designs specifications for the NICE first prototype character modules (CMs) in terms of domain information, personality information, and dialogue behaviour.

The goal of the NICE character modules (CMs) is to be capable of processing highly complex and unpredictable spoken and/or gesture user input. The functional requirements described in the present deliverable must be sufficient to generate appropriate output in context through speech, facial expression, gesture, gaze, head and body posture, body movement, and/or physical action involving objects present in the environment. Below, those requirements are described in terms of CMs' domain information, personality information and dialogue behaviour.

The present section 1 will go on to describe current assumptions as to the input and output of the character modules (see also Deliverable D6.1). These assumptions focus on the NLU and the DM (see the list of abbreviations in Section 2.2) because, except for D6.1, we are still far from having specified the other components of the input processing chain.

Section 2 presents a first HCA character module requirements and design specification. Section 3 presents the AI planning approach adopted for the fairy tale characters.

1.2 Preliminary assumptions on CM input

1.2.1 Functions producing the CM input

Goal: the functions described in this section must be capable of producing sufficient information about the user's input and the current VW state to enable the CMs to achieve their goals.

It is assumed that users' speech and gesture input is being processed by the following functions: SR, NLU, GR, GI, and IF (see the list of abbreviations in Section 2.2). This input processing chain is common to all CMs. The exact nature of the coupling between the NLU, GI, and IF functions remains a matter for further investigation.

In addition, IF output may be pre-processed by what is here preliminarily called the DM before being sent to the relevant CM. Thus, one of the DM's roles could be to decide to which CM the IF output should be sent. This decision, it seems, will require that the DM has access to the current state of the VW and hence knows what the user can perceive on the screen. Another role for the DM could be to add information about the current VW state to the information delivered by the IF. In the terminology of NICE deliverable D6.1, the DM would seem to be functionally equivalent to the Simulation Module and the Message Dispatcher.

1.2.2 CM input data structure

Here follows a high-level description of an example of the components of the IF input to the DM.

The IF will deliver a frame representation of the user's input which has the following sections, at least:

- separate sections including relevant (to the DM and the CMs) information from the SR, the NLU, the GR, the GI, and the IF, respectively;
- possibly, a high-level section on whether the input is speech-only, gesture-only, or combined speech-gesture;
- confidence scores from the SR and the NLU;
- possibly, confidence scores from the GR, the GI, and the IF;
- possibly, other event messages from the SR, such as OOV messages;
- possibly, other event messages from the NLU, the GR, the GI, and the IF;
- a section for problem messages from the NLU, such as domain/topic/value ambiguities, type mismatches, etc.;
- possibly, a section for problem messages from the GI and the IF;
- a domain section in which the domain(s) addressed by the user is ticked;
- a topic section in which the topic(s) addressed by the user is ticked;
- for the addressed topic(s) for which a topic tick is not sufficient: values conveying the more precise semantics of the user's utterance and/or gesture;
- the topic section could be sub-divided according to domain;
- the topic section might be sub-divided into a section with meta-topics which could relate to any domain and topic, and a section with domain and topic-specific information;
- the meta-topic section might be further sub-divided in some useful way;
- a speech acts section classifying the user's input in terms of speech acts at a level sufficient for determining who has the initiative at any point in the conversation;
- possibly other sections which will not be used by the SR/NLU/GR/GI/IF but will be used later by the DM or by the CM modules.

To the above frame information from the IF, the DM will add:

- information on which character is being addressed by the user;
- information on the current state of the VW.

1.3 Preliminary assumptions on CM output

The CMs will send their output to the RG. The RG will split this input into TTS output and graphics rendering instructions.

CM output will be a frame containing semantic output instructions for speech output and graphics rendering.

2. First HCA Character Module Requirements and Design Specification

2.1. Introduction

This section describes functional requirements to the HCA CM. Compared to the fairy tale CMs, the HCA CM will:

- be more "intellectual" or "brainy", performing more domain-oriented linguistic input-output processing and hence more complex internal (CM) processing of the user's spoken input;
- be less action-oriented and hence doing less AI planning.

2.2 List of abbreviations

CF = communicative function module
CG = communicative goal
CHi = conversation history
CIP = communicative intention planner
CM = character module
CMM = character module manager
DA_life = domain agent life
DA_meta = domain agent meta-domain
DA_pres = domain agent presence
DA_user = domain agent
DA_VW = domain agent virtual world
DA_works = domain agent works
DM = dialogue manager
GI = gesture interpreter
GR = gesture recogniser
HCA = Hans Christian Andersen
IF = input fusion
KB = knowledge base
MSA = mind state agent
MSAM = mind state agent manager
NLU = natural language understander
NCA = non-communicative action module
OOD = out-of-domain (input)
OOT = out-of-topic (input)
OOV = out-of-vocabulary (input)
RG = response generator
RST = rhetorical structure theory

SR = speech recogniser
TTS = text-to-speech
UM = user modelling module
VW = virtual world (HCA's study and the fairy tale world)

2.3. Domain information for the HCA CM

2.3.1 Domain list for HCA

The goal of the domain list is to circumscribe HCA's knowledge (or potential domain of discourse) at a high level of abstraction. Thus, the following domain list is a list of the general domains about which the HCA CM will be able to have conversation with the user. In this context, user are primarily (i) children and adolescents, and secondarily (ii) adults.

1. HCA's life;
2. HCA's fairy tales;
3. HCA's person and perceived (on the screen) physical presence in his study, the study itself, and the objects which are present in the study, some of which may become perceivable only if and when HCA shows them to the user. Objects will include tangible objects which the user can address through gesture;
4. HCA's role as "gate keeper" for access to the fairy tale games world which can be accessed through a door in his study. This role may be augmented with HCA's knowledge of what presently happens in the fairy tale games world as well as with possible roles for HCA as co-actor with the characters in that world.
5. The user. HCA will address this domain by asking questions about the user. He will store the information and use it during conversation.

2.3.2 Topics list for HCA

The term *topic* refers to a component, or sub-domain, of a domain of discourse. Thus, each of HCA's domains of discourse are sub-divided into a set of topics. This set, then, extensionally defines a domain at a more operational level. The notion of a topic provides a convenient way of structuring HCA's knowledge base (KB) and output repertoire. In addition, this notion may provide some leverage for input prediction, facilitating processing of the user's subsequent input turn.

Obviously, the topics lists below will grow and change as we gather and analyse data from actual conversations between a simulated HCA and users.

2.3.2.1 HCA's life

1. Start from scratch (for users without prior HCA knowledge)
2. HCA and his childhood in Odense
   - 2.1. Birth/baptism/confirmation
   - 2.2. Childhood/family
   - 2.3. Home
   - 2.4. Friends/enemies
   - 2.5. School
   - 2.6. Spare time
3. HCA and his youth in Copenhagen

    3.1. Going to Copenhagen
    3.2. Getting a foothold in Copenhagen
    3.3. More education
    3.4. First literary attempts and breakthrough
4. HCA and the women
    4.1. Riborg Voigt
    4.2. Jenny Lind
    4.3. Other women (or men?)
5. HCA and the travels
    5.1. General questions
    5.2. Country-specific questions
6. HCA and his peers
    6.1. General questions
    6.2. Name-specific questions
7. HCA and the inventions at his time
    7.1. General invention questions
    7.2. Invention-specific questions
8. HCA and Odense at his time
    8.1. Odense when HCA lived there
    8.2. Odense after HCA moved away
9. HCA and Copenhagen at his time
10. HCA and Denmark at his time
11. HCA and how he was viewed at his time (including wealth and fame)
12. HCA in historical context (events like wars etc.)
13. HCA and the present view on him (including his diaries)
14. HCA and anachronisms (e.g. inventions after his death till 2005)
15. HCA in general (e.g.: tell me about yourself, HCA / care for chewing gum?)
16. HCA and his professional knowledge (general related knowledge that may be of interest to the primary target group, e.g. interpretation methods)
17. HCA and his dreams (including superstition, dangerous and mystic things – (lots of dream material in Danish on the HCA Centre web site)
18. Denmark today (including Odense and Copenhagen)

2.3.2.2 HCA's fairy tales (works)

1. "There Is No Doubt About It"
2. Holger Danske
3. Jack the Dullard
4. Little Claus and Big Claus
5. Little Tiny or Thumbelina
6. Little Tuk
7. Lucky Peer
8. Ole-Luk-Oie, the Dream-God

9. She Was Good for Nothing
10. The Brave Tin Soldier
11. The Butterfly
12. The Darning-Needle
13. The Dryad
14. The Elfin Hill
15. The Emperor's New Suit
16. The Fir Tree
17. The Flying Trunk
18. The Goblin and the Huckster
19. The Goloshes of Fortune
20. The Little Match-Seller
21. The Little Mermaid
22. The Marsh King's Daughter
23. The Nightingale
24. The Philosopher's Stone
25. The Princess and the Pea
26. The Psyche
27. The Red Shoes
28. The Shadow
29. The Shepherdess and the Sweep
30. The Snail and the Rose-Tree
31. The Snow Man
32. The Snow Queen
33. The Story of a Mother
34. The Swineherd
35. The Tinder-Box
36. The Toad
37. The Top and Ball
38. The Travelling Companion
39. The Ugly Duckling
40. The Wild Swans
41. What the Old Man Does Is Always Right

2.3.2.3 HCA's person and physical presence

1. Physical appearance
    1.1. Looks
    1.2. Clothes
2. Physical behaviour
    2.1. Habits
    2.2. Actions

3. Study
    3.1. Interior
    3.2. Objects

2.3.2.4 HCA's role as "gate keeper"

1. Game invitation
2. Game information

2.3.2.5 Questions to the user

1. Personal questions
    1.1. Name
    1.2. Age
    1.3. Sex
    1.4. School
    1.5. Origins
    1.6. Friends
    1.7. Hobbies
    1.8. Girlfriends/boyfriends/married
    1.9. Travels
2. Favourite fairy tales

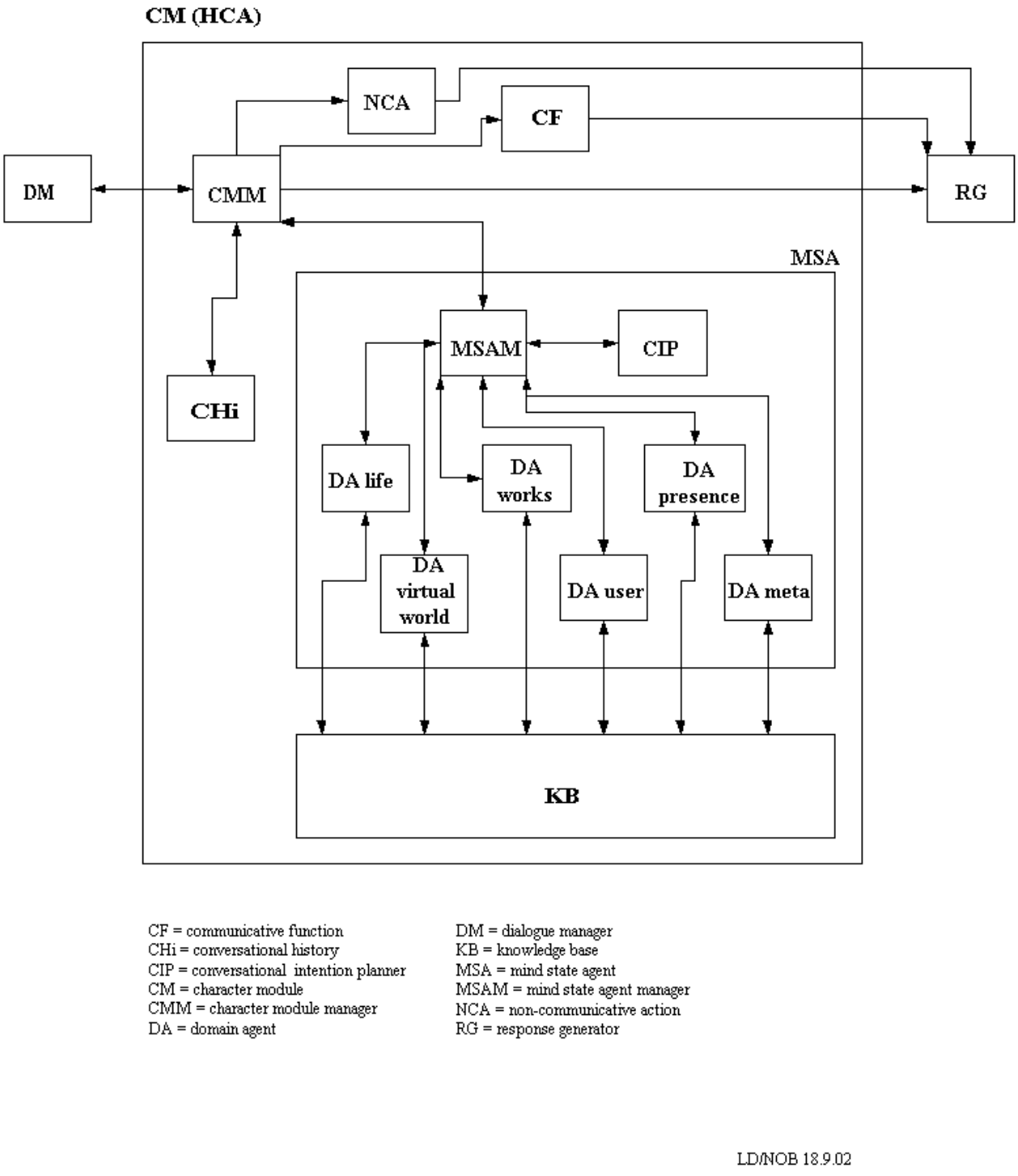


Figure 1. Draft high-level architecture for the HCA character module.

2.3.3 Implementation of HCA domains and topics (KB and NLU)

HCA's domains and topics will be implemented in a relational database (or knowledge base, KB), such as Access of FoxPro, for ease and speed of access and complex querying. The main KB structure will be a hierarchy of domains, topics, possibly sub-topics, and output semantics.

Domain, topic, sub-topic, and semantic representations will have preconditions of use, enabling the MSA to retrieve the right ones for any communicative goal.

The NLU will combine state-of-the-art robust island parsing, anaphor resolution, and a statistical topic spotter based on the data gathered on HCA-user conversations. The NLU will share domain, topic, sub-topic, and semantics indexing structure with the KB, enabling fast matches between the user's processed input and HCA's repertoire of responses.

The implementation strategy for the first prototype is to enable users to address all five HCA domains of discourse in order to gain early experience with the challenges imposed by each and every one of these. This, rather demanding, early implementation strategy will be counter-balanced by only selecting a fragment of the intended domain topics (see 3.2) for early implementation. Similarly, the first-prototype HCA will have less articulate emotional states than in the second type, and probably also a less complex communicative goal structure (see below).

2.3.4 HCA architecture

Figure 1 shows the draft architecture of the HCA CM.

2.4. Personality information for the HCA CM

HCA's personality is a main factor in shaping and determining the discourse. At this point, the following is known about his personality. HCA:

- is lively, quick-witted and imaginative, like Jack the Dullard;
- often takes the initiative during conversation;
- is emotionally sensitive;
- has a strong visual and personal presence;
- can be quirky at times, for instance refusing to discuss certain topics.

The personality style just described means that HCA's personality is a central input control factor. Whenever HCA takes the initiative during the dialogue, we may expect the user to respond in kind. This makes prediction of the next user input much easier and hence also makes its processing easier.

In appearance, HCA will be in his 50s, as life-like as possible, his physical appearance being based on photographs and paintings of him.

It should be noted that HCA's personality will not be implemented as such, i.e. the NICE system will not include an acting software module which implements HCA's personality characteristics (except for the visual ones). Rather, his personality will be reflected in the operations of the HCA CM and hence ultimately in his conversational behaviour.

2.5. Dialogue behaviour of the HCA CM

2.5.1 Goal

The goal of the HCA CM is to process the input from the DM module and send a semantic representation to the RG which will result in the presentation to the user of appropriate output in context. To be appropriate, the output should not only adequately address the user's input but also consistently reflect HCA's personality as described above.

2.5.2 Functionality and modularity of mainstream conversation processing

The HCA CM will include the following functions or modules, at least (cf. Figure 1):

- a character module manager (CMM);
- an active conversation history (CHi);
- a mind state agent (MSA), including:
  - a mind state agent manager;
  - a conversational intention planner (CIP);
  - six domain agents (DAs), one for each of HCA's five domains of discourse and one for meta-domain issues;
- a knowledge base;
- a communicative function module (see 5.3);
- a non-communicative action module (see 5.4).

This section describes the modules in more detail, focusing on their functional specification.

2.5.2.1 Character module manager

The HCA CM will include a character module manager (CMM) which manages the overall data flow between the internal agent modules as well as with the DM and the RG.

An important goal in CMM design and construction is to make the CMM as devoid of particular domain and topic contents as possible, so that the CMM can be re-used for other domains, topics, and characters with little or no-re-implementation.

The CMM receives from the DM a semantic representation of the user's input in the form of a user input frame. The CMM may do some pre-processing of the frame before passing it on to the CHi, (later) to the MSA, and ultimately to the RG. This will be determined through use cases informed by incremental NLU, GI, IF, DM, and CM design.

In addition to receiving frames from the DM, the CMM will also receive data structures from the DM which will be passed on, without or with pre-processing, to the CHi and either the CF or the NCA (see below).

In summary, the CMM will be in one in three different main states, either (i) processing domain and topic-related information (or communication contents) for use by the CHi and the MSA, (ii) processing fast track communicative function information for use by the CHi and the CF, or (iii) processing non-communicative information for use by the CHi and the NCA.

In the following sections 5.2.2 through 5.2.7, we focus on the CM's processing of communication contents as opposed to communicative function information.

2.5.2.2 Conversation history

Given the DM input frame, the CMM first consults the conversation history (CHi).

The goal of the CHi is to store and pass on such information on the discourse context as will be sufficient for the subsequent CM modules to accomplish their joint task of providing appropriate and entertaining output to the user in any circumstance.

An important goal in CHi design and construction is to make the CHi devoid of particular domain and topic-related processing, so that the CHi can be re-used for other domains, topics, and characters with little or no-re-implementation.

The CHi is a recording agent which incrementally builds a record of the conversation (the discourse context) in terms of the frame attribute-value structures, such as confidence score, domain, topic, semantics, speech acts, modalities used, current VW state, and others (see below). This record includes all of the user's input as well as all of HCA's output.

Based on the conversation record, the CHi adds to the current frame its discourse context in terms of:

(i) annotations based on the attribute-value structures which are shared by the input frame and the CHi's discourse context representation, such as same_domain, domain_shift, same_topic, topic_shift, same_modality, modality_shift, continued_problems, continued_problems_only, etc.;

(ii) higher-order discourse structure representations, such as who takes, gives, or preserves the initiative in conversation, possibly RST structures and others as well. Such additional structural representations will be developed as a result of user-HCA conversation data analysis. Primarily, they will be built upon the foundation of the domain, topic, semantics, and speech acts attribute-value structures provided in the frames from the DM. Secondarily, it cannot be excluded at this stage that new CHi structures will impose additional demands upon the output from the DM and their predecessor modules;

(iii) turn numbers for user and system turns;

(iv) HCA's emotional state as represented by attribute-value structures. At initialisation time, the CHi will include HCA's default emotional state which will be passed on to the MSA. Subsequent modifications to HCA's emotional state will be reflected in subsequent frame annotations stored in the CHi. In the first prototype, HCA will have a small set of emotion attributes, such as happiness and anger, each of which will have a small set of possible values (strengths). HCA's default emotional state will reflect our emerging conception of HCA's personality. This state will, or may, change as a function of the discourse context. The values constitute a source of input to HCA's current communicative goal formation. Thus, for instance, if HCA gets angry with the user because of the user's input in context, he may take the initiative in order to change the topic of conversation altogether.

The CHi returns the discourse context-augmented frame to the CMM.

2.5.2.3 Mind state agent

The CMM passes on the augmented frame from the CHi to the mind state agent (MSA) and, more particularly, to the mind state agent manager (MSAM).

An important goal in the design and construction of the MSA is to make its architecture re-usable without modification for implementing conversational characters other than HCA.

The MSA represents HCA's current mind state as informed by our knowledge of, and design decisions on, HCA's personality. Thus, the MSA will be a running software embodiment of HCA's personality as this personality manifests itself as a function of the user's input. The MSA will decide to modify, or not to modify, HCA's current mind state (emotions and communicative goals) as a function of the user's input. This will be determined by modification rules applied by the MSA, rules which have been made at design-time in order to reflect our (the designers') conception of HCA's personality. In this way, those rules embody the "causal" influence of a virtual (non-implemented) HCA personality model, as if the system included such a personality model which acted on the user's input. Given the user's input, the MSA will determine whether and how to modify HCA's current mind state by applying emotion modification rules and communicative intention modification rules. For instance, given a certain input in context, the MSA may:

- change emotion_value from happiness to anger
- change CG_value from continue_domain to change_domain
- change CG_value from preserve_initiative to change_initiative

- etc.

2.5.2.4 Mind state agent manager

The goal of the MSAM is to manage the overall data flow between the MSA modules as well as to exchange information with the CMM and the KB.

An important goal in MSAM design and construction is to make the MSAM as devoid of particular domain and topic related information as possible, so that the MSAM can be re-used for other domains, topics, and characters with little or no-re-implementation.

The MSAM may perform some pre-processing of the frame before it is passed on to the CIP.

2.5.2.5 Communicative intention planner

The communicative intention planner (CIP) represents HCA's communicative goals and annotates the frame accordingly before it is passed on to the domain agents.

By default, HCA has a single top communicative goal (or intention). It is to guide the user through HCA's five domains of knowledge. However, even this top communicative goal may be overridden when HCA gets carried away in some exceptional emotional state. In all other cases, his emotional state serves to temper his output in certain ways whilst he continues to adhere to his top communicative goal.

For every user, HCA will have as his top goal to test and entertainingly satisfy the user's interest in his five knowledge domains, i.e.:

1. HCA's life;
2. HCA's fairy tales;
3. HCA's person and perceived (on the screen) physical presence in his study, the study itself, and the objects which are present in the study, some of which may become perceivable only if and when HCA shows them to the user. Objects will include tangible objects which the user can address through gesture;
4. HCA's role as "gate keeper" for access to the fairy tale games world which can be accessed through a door in his study. This role might be augmented with HCA's knowledge of what presently happens in the fairy tale games world as well as with possible roles for HCA as co-actor with the characters in that world.
5. The user. HCA will address this domain by asking questions about the user.

HCA's top goal thus implies five sub-goals which he wants to achieve. As the user has the initiative at the start of the conversation, the user will be the first to determine the domain of actual conversation. This domain may be any of the domains 1-4 above. However, during conversation, and depending, of course, on the tenacity and interest profile of the user, HCA will systematically grab the initiative in order to open the remaining domains of conversation, and he will continue with those to the extent that the user shows any interest in them. The frame attribute *domain_value* as recorded in the discourse history will be useful for implementing this functionality.

For instance, if the user shows any strong interest in HCA's life, HCA will continue conversation in that domain for a while. On the other hand, if the user does not show any particular interest in HCA's life, works, or physical surrounds, HCA will eventually offer the user to visit the fairy tale world (which is likely to attract the curiosity of most users). To systematically achieve his five domain goals as a function of the user's interest, HCA (CIP) will score the user's input to gauge the user's interest. In this way, HCA will create and build, for each user, a model of that

user's interests. In addition, through his questioning of the user, HCA will collect additional user information for use during conversation, for instance on the user's age. The CIP will include a user modelling module (UM) for these purposes.

We may not have the time in NICE to build a second stage onto the presence of user modelling functionality in the CIP. However, it could be fun to have HCA tell new users about previous users like them. In any case, we will consider enabling HCA to articulate observations on the current user, such as that it is nice to speak to someone who is that interested in HCA's fairy tales.

Secondly, as said already, HCA will have communicative goals determined by his emotional state. These goals may completely override his five sub-goal agenda. For instance, HCA may get so mad at the user that he insists on getting a particular reply in order to continue the conversation at all.

Thirdly, the CIP will have a topic-based list of communicative goals which reflect HCA's particular personality. For instance, HCA may like a particular topic so much that he will try to continue conversation on it even after the user has lost interest, or he may dislike a particular topic so much that the user will find it very hard, if not impossible, to make him talk about it. These goals will be specified as we learn more about HCA's personality from domain experts as well as about the users' interests in him from Wizard of Oz simulations. HCA (MSA) will maintain an extensible list of those goals expressed in terms of topics of conversation.

Fourthly, HCA's communicative goals will, of course, include the desire to repair problems in the user's input (OOV, OOD, OOT, noise, silence, etc.). This "repair" functionality goes beyond standard meta-communication abilities which tend to be aimed at getting the communication back on track. In the case of HCA, we also have to reckon with situations in which HCA is lost wrt. the user's input and has to use strategies more interesting than trying to figure out what the user is talking about. Thus, for instance, when faced with substantial OOV/OOD in context, HCA will have a repertoire of strategies for changing the topic of conversation.

The CIP will use planning rules, priority rules, and conflict resolution strategies for determining HCA's current conversational goals given the user's input in context.

An important issue in CIP (preliminary) goal formation is how to deal with input problems. Thus, before forming a preliminary communicative intention based on its goal structures, the CIP first checks the input frame for low input confidence, problems-only, OOVs, OODs, OOTs, other event messages, etc. (all input modalities). If any of these obtain, the CIP checks for recent repeated occurrences in the CHi record. The resulting error message annotation is passed on to the CIP's communication problems planner and goal instructions are created for the DA and the KB. The communicative intention formed by the CIP will depend on the details of the error message annotation. It will range from preserving the domain and topic, doing more or less standard repair meta-communication, to changing domain altogether, including the making of comments on the user's communicative behaviour, jokes, etc.

In the absence of major problems such as those described above, the CIP checks for semantic problems in the input, such as domain/topic/semantic value/speech act ambiguities, type mismatches, etc. (all input modalities). If any of those obtain, the CIP checks for recent repeated occurrences in the CHi record. The resulting error message annotation is passed on to the CIP's communication problems planner and goal instructions are created for the DA and the KB.

In the absence of problems such as those described above, the CIP checks the CHi record and uses its domain and topic-oriented communicative goal structures to form a preliminary

communicative intention. This preliminary communicative intention is then tested with the relevant DA and the KB (see below).

Having annotated the input frame with communicative goal information, the frame is sent back from the CIP to the MSAM. The MSAM passes on the frame to the relevant domain agent.

2.5.2.6 Domain agents

Given his five domains of conversation, the HCA CM will have six domain agents (DAs). Five of these will be responsible for the detailed handling of conversation in a single domain each, and the sixth DA will be responsible for the detailed handling of conversation in meta-domain areas, such cross-domain ambiguities, initial greetings, end greetings, etc. So, the DAs are:

- DA_life = domain agent life
- DA_meta = domain agent meta-domain
- DA_pres = domain agent presence
- DA_user = domain agent
- DA_VW = domain agent virtual world
- DA_works = domain agent works

On receiving the input frame from the MSAM, the relevant DA will first check the CHi information and the CIP information in the frame. Depending on the information, the DA will either return the frame to the MSAM or compose a query to the KB. The DA will return the frame to the MSAM without querying the KB in cases where querying is irrelevant because the DA already has procedures for solving whatever problem the frame may pose. In all other cases, the DA will compose a query to the KB. Where the cut will lie between these two generic cases, will be determined as the functional design proceeds. In both generic cases, the DA will often have to perform domain-oriented reasoning on the user's input before taking action.

On receiving the KB's return, the DA may again have to perform domain-oriented reasoning before returning the frame to the MSAM.

The core of the KB return will always be semantic representations of the output to be sent to the RG. This output will either consist in coordinated spoken and graphics output or in graphics output-only. Informally, for instance, the KB may return the following:

- S_response: S1_S: I am HCA. I write fairy tales. How old are you?
- GR_response: S1_GR: Stop writing. Look up at user. Smile. Statement_face then Question_face.

Based on the query, the KB will return suggestions as to whether to address the user through graphics-only or through a combination of speech and output graphics. In other words, the DA/KB will need to return information on both:

- what to communicate, and
- how to communicate

The *what* to communicate depends on HCA's current mind state including his communicative goals list, the user's input, and the discourse context. Among these, HCA's current mind state takes priority. Thus, HCA will continue the conversation on the present topic, unless he:

- changes goals, for instance because he has nothing, or no more, to contribute to the current topic, because he estimates that the user has lost interest in the present domain of conversation, or because he does not appreciate the present topic of conversation

- changes emotion sufficiently strongly to change goals

HCA's topic changes are determined by rules linking the input semantics, topics, domain, context, and HCA's current mind state to HCA's repertoire of topic changes.

The *how* to communicate offers seven options (modality combinations):

1. through speech-only
2. through gesture-only
3. through facial-only
4. through speech and gesture
5. through speech and facial
6. through gesture and facial
7. through speech and gesture and facial

Note that gesture includes object manipulation.

In addition, the KB will return instructions on how HCA's emotional state should be modified as a result of the user's input, who has the initiative as a result of HCA's output, which speech acts are included in HCA's output, which domain and topic are being addressed, whether a domain and/or topic change has been made, etc.

The results of the DA's processing of the input frame will be sent back through the MSAM and the CIP to the CMM. The CMM will pass on the resulting frame to the CHi and the RG.

The CHi will store HCA's output as part of its incremental construction of the discourse context, and the stored information will then be added to the next user input frame. In this process, it is essential to the preservation of the domain-independence of the CHi that the CHi can store HCA's output in a completely mechanical manner, without having to apply domain-oriented knowledge. For this to be possible, the KB return will have to be pre-processed on its way back to the CHi. Whilst this is not rocket science to do, it will require quite some care and common sense to make sure that the pre-processing steps be optimised from an engineering point of view. Solving this problem requires careful functional design of the DAs, the MSAM, and the CIP and its UM. This is work in progress.

2.5.2.7 Knowledge base

The knowledge base (KB) will be a powerful relational database which will be indexed according to domain, topic, possibly sub-topic, and semantics in a way which is isomorphic to the organisation of the NLU (and possibly the GI and IF as well). In this way, the KB will be well-suited for communicating with the individual DAs. The KB will interpret DA queries and incrementally tick off returns already produced in order to keep the DAs informed about output which should not be repeated. Query interpretation will include interpretation of CIP communicative goal instructions on whether to change domain, topic, initiative, output modalities, etc. The KB may include lists of alternative responses to a given query to be selected at random in order to make HCA full of surprises as a partner in conversation.

2.5.3 Communicative functions

In addition to mainstream conversation processing, the HCA CM will include functionality for processing communicative functions (CF). Communicative functions are here considered to be elements of communicative behaviour which people exhibit when they are being addressed during conversation. For instance, when a new user comes up to HCA and starts communicating with him in a situation in which HCA is busy writing a new fairy tale, HCA should immediately

show signs of acknowledging that he is being addressed even prior to responding to the user. He might, for instance, look up towards the user and stop writing without putting down his pen. By definition, such communicative behaviour must be initiated before the user has stopped speaking and hence cannot be produced as part of mainstream conversation processing.

The HCA CM will include a fast-track module, the CF, for producing HCA's communicative function behaviours whilst listening to the user. The CF will be prompted by the CMM. These feedback behaviours will be selected from a repertoire of listening behaviours and sent to the RG for execution. The CF feedback will be various combinations of facial expression, head and body movement, gesture and speech (e.g. back-channelling).

2.5.4 Non-communicative action

The CMM will also be responsible for ensuring that HCA remains visually active when he is not being addressed by anyone. When this is the case, the CMM will instruct the non-communicative action module (NCA) to take over, selecting HCA's actions from a repertoire of behaviours and sending their semantic representation to the RG for execution.

2.5.5 Data flow

The main data flows in the HCA CM are apparent from the description in this report as supported by Figure 1.

3. First Fairy Tale Character Module Requirements and Design Specification

3.1. Introduction

As pointed out above, the fairy tale characters are expected to perform much more physical action than HCA. This section presents an AI planning formalism for specifying the actions of the fairy tale characters. Obviously, the same formalism can be used for specifying the, comparatively fewer, physical actions to be performed by HCA.

3.2 A simple scenario

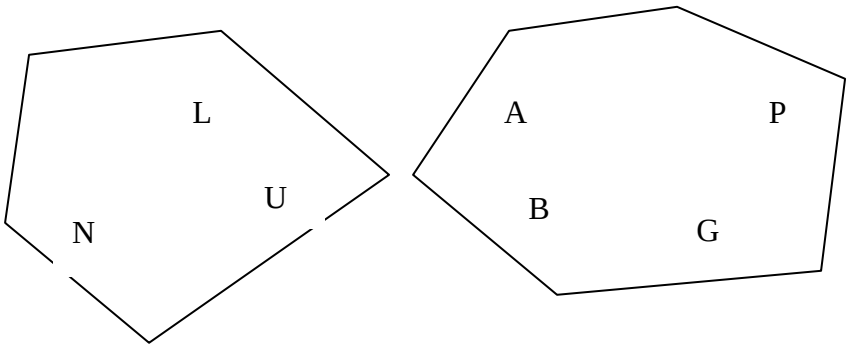
The actors:

- a princess P
- two suitors, A and B
- the user's avatar U

The objects:

- an artificial nightingale N
- a ladder L
- a gold coffin G

The world:



The world is divided into two areas by a deep shaft, which can only be bridged by the ladder. The two suitors want to marry the princess. The goal of the princess is to obtain the artificial nightingale. Unfortunately, she is confined to the castle (why? because she is too lazy?), so she will marry the man who brings it to her. The suitors obviously need the help of the user to get the nightingale, either by getting the user to fetch the nightingale, or by getting the user to place the ladder over the shaft.

The user has a choice whether to help any of the suitors (or perhaps try to marry the princess himself!). The user can try to get the suitors explain why they need help, and try to get something in return(?).

An example dialogue that might take place:

A: Could you please fetch the artificial nightingale for me?
U: Why?
A: Because I need to give it to the princess.
U: Why do you want to do that?
A: Because I want to marry her.
U: I see. Where is the nightingale?
A: It is in the hollow tree.

In a later, more advanced version of the system, the dialogue might go:

U: If I get you the nightingale, I want some gold in return.
A: Ok, I'll get it for you.

The scenario, simple as it is, has some interesting features. The two suitors have to plan a complex chain of actions in order to fulfill their goals. Attaining the goal requires interaction with the user and with the other characters. The two suitors will have to compete with each other to obtain restricted resources (the gold coffin, the help of the user...). The dialogue also contains some interesting elements, like utterances of the kind "I will do as you say, provided you do this" which might have far-reaching consequences for the behaviour of the other characters.

In order to create a longer fairy-tale-like history, the princess can present yet another task for the suitor when he brings the nightingale.

This set-up allows for a flexible interactive story to unravel. Although the actors have limited capabilities, there are still a number of possible paths the story can take. The user can influence the outcome.

3.2.1 Coding the scenario

Our basic assumption is that descriptions of scenarios like the one above will be coded in a rule-based formalism, rather than coded in a conventional programming language. The reasons for this are:

- Rules will be used both for planning (by agents) and execution (by the simulation system). Thus specifications can be reused by different modules, which enforces consistency throughout the system.
- A rule-based system is non-deterministic; i.e. it describes an infinite number of possible developments in a concise way.
- A rule-based system is incremental, i.e. rules can be added and removed while the system is running.

The purpose of this section is to outline such a formalism, the working name of which is OOPEL (Object-Oriented Planning and Execution Language). The formalism is object-oriented because

- it allows for reusable specifications and inheritance,
- it allows agents to plan from a subjective point of view (each agent plans for himself). The significance of this will be made clear later in the text.

A position in the world is linked to other positions in two different ways. First of all, a position *p* has a set of *adjacent* positions, i.e. positions an agent can move to if its current position is *p*.

Secondly, *p* has a set of attainable positions. An agent can manipulate objects on *p´*s attainable positions if the current position of the agent is *p*. Movable objects (such as the ladder) and agents (such as the princess) have that in common that they are sitting at a specific position. However, only agents can carry (movable) things and perform actions. This state of affairs is described as follows:

```
class Position {
  constant set of Position adjacent;
  constant set of Position attainable;
};

class DynamicObject {
  Position sitting_at;
}

class Movable extends DynamicObject {}

class Agent extends DynamicObject {
  set of Movable carrying;
  < action definitions >
 }
```

An action typically has both pre-conditions and effects. The preconditions are conditions on the world state that have to be fulfilled before the action can be carried out. The effects describe properties of the world state after the action has been performed. For instance: It is possible for an agent to pick up a movable object if the object is attainable. The effects of the "pick up" action are that the agent is carrying the object, and that the object is no longer sitting at its orginial position (we assume that an object is "sitting at " a position only if it is available for agents to pick up). The complete action is coded as follows:

```
  action pickup( Movable m, Position p )
    possible when
      (m.sitting_at == p) && (sitting_at.attainable == p)
    results in
      (carrying >> m) && (m.sitting_at != p);
```

We assume here that the symbol ">>" denotes set membership.

Actions can also be overloaded in the same way methods are overloaded in Java. For instance, heavy movable objects might require special resources:

```
  action pickup(HeavyMovable m, Position p)
    possible when
      (m.sitting_at == p) &&
      (sitting_at.attainable >> p) &&
```

```
      (strength>100)
    results in
      (carrying >> m && m.sitting_at != p);
```

The inheritance mechanism allows action definitions to be overridden, as follows:

```
  class Princess extends Agent {
   action pickup(HeavyMovable m, Position p)
     possible when false;
 }
```

In the example above, a Princess is just like an Agent, with the exception that she will never pick up a heavy object (since her definition of the action "pick up" for HeavyMovables overrides the corresponding action for general Agents).

Actions that involve more than one agent have to be synchronized. For instance, if agent A wants to give B an object, it is important that B receives the object in order for the operation to succeed. For simplicity, such a complex transfer operation is coded as two simpler operations that have to follow one another.

```
action give( Agent a, Movable m )
  possible when
    (carrying >> m)
  to be followed by
    a.receive(this, m);

action receive( Agent a, Movable m )
  must follow
    a.give(this, m)
  results in
    (carrying >> m && a.carrying !> m);
```

We assume here, similarly to Java, that "`this`" denotes the object performing the action, and that "!>" means "does not contain". The first action declaration above says that in order for me to give object m to agent a, I have to carry m, and the "give" action has to be directly succeeded by a corresponding "receive" action from agent a. The second declaration says that in order for me to receive object m from agent a, the "receive" action has to be preceded by a corresponding "give" action from agent a.

In the scenario, the princess will marry the suitor that presents her the nightingale. The act of accepting a proposal of marriage is coded as follows: the princess has a certain movable resource (her hand) which she gives to the suitor whom she intends to marry. To code the condition "I will only marry you if you give me the nightingale", the princess has a special "give" action for her hand.

```
  class HandofPrincess extends Movable {}
```

```
class Princess extends Agent {
  action give( Agent a, HandOfPrincess h )
    possible when
      carrying >> h
    must follow
      receive(a, Nightingale)
    to be followed by
      a.receive(this, h);
}
```

The "must follow" statement in the action declaration above asserts that a princess can only give her hand to an agent *a* if the agent *a* first have given a nightingale to the princess. In this statement, "`Nightingale` " is to be interpreted as an existentially quantified variable of type Nightingale.

3.2.2 Goals

Each agent has one or more goals it is trying to fulfill. The goals are expressed as properties of the desired world state. Initially, the princess has the goal

```
carrying >> Nightingale
```

and the two suitors both have the goal

```
carrying >> HandOfPrincess
```

As the game evolves, the goals of the agents will change as a result of incoming stimuli or interaction with other characters, or because agents lose interest or find the goals unsatisfiable.

3.2.3 Planning and execution

In order to fulfill a goal *g*, an agent must find a sequence of actions that transforms the current world state into a state where *g* is true. This process is known as *(AI) planning* and is a well-researched area. Note, however, that in this setting the problem differs from the classical formulation of the problem in several regards:

- The world is changing over time, since other agents move about in the world and influence its state. This means that a chain of actions that seemed possible at planning time might turn out to be impossible to perform.
- Each agent has only incomplete information about the state of the world.
- Some tasks require the collaboration of more than one agent (like marrying); still each agent can only control its own actions.
- The outcome of certain actions is uncertain, e.g. asking another agent for help does not mean he will comply.

Because of these additional difficulties, the agents' reasoning capabilities will not consist of a planner alone. Rather a planner will be used as a subsystem, but the plans it produces will

constantly be assessed, and sometimes be rewritten or discarded. The agents must decide which goals to pursue next (i.e. which plans to construct next).

As an example how to cope with the last two difficulties, we consider the suitor's problem of acquiring the hand of the princess. The first plan constructed by suitor is simple: After the suitor has walked up to the princess, the princess gives her hand, the suitor receives it, and has thus achieved his goal.

```
princess.give( suitor, HandOfPrincess )
receive( princess, HandOfPrincess )
G: carrying >> HandOfPrincess
```

(Lines beginning with "G:" denote achieved goals). However, this is not an executable plan for the suitor, since one of the actions can only be performed by the princess. Not knowing the goals of the princess, the suitor will therefore add a "request" action to his plan, asking the princess to give him her hand.

```
request( princess, princess.give( suitor, HandOfPrincess ) )
princess.give( suitor, HandOfPrincess )
receive( princess, HandOfPrincess )
G: carrying >> HandOfPrincess
```

To check the possibility of performing the "give" action, the princess has to check the conditions of her "give" rule.

```
action give( Agent a, HandOfPrincess h )
  possible when
    carrying >> h
  must follow
    receive(a, Nightingale)
  to be followed by
    a.receive(this, h);
```

She notes that one of the conditions are not fulfilled: She has not performed a "receive" operation, getting the nightingale from the suitor. Thus, instead of performing the operation, she sends her "give" action rule back to the suitor. The suitor then has to incorporate that into his plan:

```
princess.receive( suitor, Nightingale )
princess.give( suitor, HandOfPrincess )
receive( princess, HandOfPrincess )
G: carrying >> HandOfPrincess
```

Now the suitor realises that he must give the nightingale to the princess, which means that he must first have the nightingale in his possession. Thus the suitor must continue to construct a (lenghty) plan how to acquire the nighingale.

The example above shows some interesting features of the proposed system:

- the agents can update each other with new knowledge. In the example, the princess sent her "give" rule to the suitor, thus informing the suitor how the princess is reasoning. The suitor can then use this new information in his own planning process.
- the agents can bargain ("you get my hand if you get me the nightingale") using a kind of lockstep protocol.
- the agents will understand when they have to communicate with other: If the performer of one of the actions on their plan is another agent, then this agent has to be contacted.

3.2.4 Long-term goals

We distinguish between (short-term) *goals* and *long-term goals*. Goals are properties of desired world states which an agent seeks to attain. After fulfillment, the agent removes the goal from its agenda and forgets about it. *Long-term goals*, on the other hand, represent behaviours that are persistent and thus contribute to the agent's personality. Long-term goals are represented as rules that add (short-term) goals to the agent's agenda as a reaction to incoming stimuli.

```
class GreedyAgent extends Agent {

  // A greedy agent always tries to get money
  // whenever possible.
  reaction get_money(Valuable v)
    when
      near(sitting_at, v.sitting_at)
    add_goal
      carrying >> v;

};
```

Intuitively, this rule says that whenever the agent is near a valuable object *v*, the agent should add a goal to acquire *v* (unless the goal is already on the agenda). The definition of the "near" predicate is given outside this formalism; it could be a boolean Java method, a Prolog predicate, or something else.

3.2.5 Coding example

As an example we define a simplified version of the parts of the class hierarchy in the scenario, which summarises the discussion so far.

```
class Position {
  // All adjacent positions, these are constant because the
  // shape of the world does not change.
  constant set of Position adjacent;
```

```
  // Objects that are on the "attainable" positions
  // can be manipulated by an agent that is on this position
  constant set of Position attainable;
}

class Dynamic_object {
  // The current position of the dynamic object
  Position sitting_at;
}

// A Movable can be moved about
class Movable extends Dynamic_object {
}

// Nightingale
class Nightingale extends Movable {
}

// HandOfPrincess: Whoever has the hand of the princess
// is effectively married to the princess
class HandOfPrincess extends Movable {
}

// An Agent is an autonomous creature that can move about,
// pick up, and put down things.
class Agent extends Dynamic_object {

  // The objects the agent is carrying
  set of Movable carrying;

  // The agent can move to an adjacent position
  action moveto(Position p)
    possible when
      (sitting_at.adjacent >> p)
    results in
      (sitting_at == p);

  // The agent can pick up an object from a position
  // attainable from the current position
  action pickup( Movable m, Position p )
    possible when
      (m.sitting_at == p) &&
      (sitting_at.attainable >> p)
    results in
```

```
      (carrying >> m) && (m.sitting_at != p);

  // The agent can put down an object at a position
  // attainable from the current position
  action putdown( Movable m, Position p )
    possible when
      (carrying >> m) &&
      (sitting_at.attainable >> p)
    results in
      (m.sitting_at == p) && (carrying !> m);

  // Give a movable object to another agent
  action give( Agent a, Movable m )
    possible when
      (carrying >> m) && (sitting_at == a.sitting_at)
    to be followed by
      a.receive(this, m);

  // Receive a movable object from another agent
  action receive( Agent a, Movable m )
    must follow
      a.give(this, m)
    results in
      (carrying >> m && a.carrying !> m);
}

class Princess extends Agent {

  // The special "give" rule of the princess, which
  // only applies to giving her hand away (i.e. marrying her)
  action give( Agent a, HandOfPrincess h )
    possible when
      carrying >> h
    must follow
      receive(a, Nightingale)
    to be followed by
      a.receive(this, h);

}
```

3.2.6 Dialogue behaviour encoding

Speech acts are actions that has preconditions and effects, just as actions like *move* and *pickup*. It is perfectly possible to model the whole dialogue decision making process as a planning process. As an example, consider the speech act of inquiring for a specific of information, like "Where

can I find a nightingale", an utterance the suitor might very well want to ask the princess in our scenario. This might be represented as follows:

```
suitor.ask_for_info( princess, ?p(Nightingale.position == p) )
```

A reaction rule describes how an agent (for instance, the princess) will react to this speech act (where the formal parameter "Agent a" then corresponds to the suitor):

```
reaction receive_ask_for_info( Agent a, Question q )
  on
    a.ask_for_info( this, q )
  results in
    a.asked == q
  add goal
    question_answered( a, q );
```

That is, an incoming info question results in the addition of a goal to answer this question. The speech act "ask_for_info" itself might be coded as follows:

```
action ask_for_info( Agent a,Question q )
  possible when
    talking_to == a
  results in
    a.receive_ask_for_info( this, q );
```

This rule says that, when asked a question, an agent will react according to the reaction "receive_ask_for_info". Thus, unless it has information to the contrary, the agent asking the question (e.g. the suitor) will assume that the recipient (e.g. the princess) is equipped with the same rule as himself and that she will hence try to answer the question.

To answer info questions, we also need a speech act "answer_with_info", which can be coded as follows:

```
action answer_with_info( Agent a, Reply r )
  possible when
    (talking_to == a && a.asked == q && r.type == success)
  results in
    question_answered(a, q)
  where
    r = response_generation( ask_for_info(a, q) );
```

Here we assume that "response_generation" is a procedure coded outside the OOPEL formalism, returning a reply containing the exact wording and the type ("success" if the question has been answered, and "failed" otherwise).

As an example how these rules might play out, we continue the nightingale example from the suitor's perspective. First the suitor asks the princess for the location of a nightingale:

```
suitor.ask_for_info( princess, ?p(Nightingale.position == p) )
```

This results in the suitor's assumption that the princess will react according to the "receive_ask_for_info" scheme:

```
princess.receive_ask_for_info(suitor, ?p(Nightingale.position == p))
```

that is, the suitor assumes that the princess has noted the question:

```
suitor.asked(?p(Nightingale.position == p))
```

and will try to answer it by adding the goal:

```
princess.question_answered( suitor, ?p(Nightingale.position == p))
```