Conceptual Discrepancies and Feedback in Human-Computer Interaction

Robbert-Jan Beun and Rogier M. van Eijk

Center for Content and Knowledge Engineering
Information and Computing Sciences
Utrecht University, P.O. Box 80089
3508 TB Utrecht, The Netherlands
{rj,rogier}@cs.uu.nl

ABSTRACT

In this paper, we present a computational framework for the detection of conceptual discrepancies in human-computer interaction. The framework is developed as a basis for the generation of feedback utterances at the ontological level. In our method, presuppositions are extracted from the user's message, expressed in a common vocabulary, and compared with the system's ontology, which is expressed in type theory. Discrepancies are detected by the system if it notices type conflicts, particular inconsistencies or ontological gaps. Depending on the kind of discrepancy, the system generates a particular feedback message in order to establish alignment of its private ontology with the mental model of the user.

1. INTRODUCTION

Contemporary technological developments of interactive systems and the expansion of bandwidth enable designers to incorporate a variety of media and modalities in the computer interface. But merely adding amazing technological feats or increasing bandwidth does not necessarily improve the communication process. When we interact with computers, we also want them to be endowed with characteristics that closely mimic human communication. One of these characteristics is the ability of humans to react in a cooperative manner to the communicative actions of the dialogue partner. In everyday conversation, people effortlessly answer questions, accept or deny assertions, confirm the receipt of a message and provide relevant feedback in case of communication problems. Since the cognitive and communicative abilities of humans are so well adapted to the real-time processing of these various interaction structures, we expect that including natural conversational skills in interfaces may contribute to a more efficient and satisfactory human-computer interaction.

One of the prerequisites for natural human communication is that participants are able to reason about and to discuss various aspects of the domain of discourse. In order to cope with the complexity of the world around us, people consider the existence of objects, discuss possible behaviour, draw conclusions from the various dialogue contributions, discuss the meaning of the communication symbols and many more. Behind these manifestations of various beliefs and opinions is the participant's need to conceptualise the problem domain and to build a coherent and consistent mental model to achieve some sort of common understanding of our complex world.

The situation in human-computer interaction hardly differs from the communicative situation in the real world. In a computer domain, for instance, a user should know that there are things like files and folders, that files are deletable, readable, editable, savable, etc. Although the need to discuss these various aspects of the virtual domain is probably even more compelling than in the real world, these conversational skills are usually absent from the computer interface.

The goal of this paper is to analyse some of the basic system requirements that pertain directly to the above natural conversational skills. For that, we will present the main building blocks and a computational method that enables us to generate feedback utterances that regulate the repair of conceptual discrepancies between a computer system and its user. Two phases will be distinguished in the repair process: first, the detection of discrepancies on the basis of particular information in the incoming message and second, the resolution of discrepancies by means of an adequate conversational strategy.

2. DETECTION OF CONCEPTUAL DISCREPANCIES

Users of a particular application or system build their own conceptualization of the characteristics and the behaviour of the application; these conceptualizations are usually termed mental models [5]. Mental models are based on the user's previous knowledge, and also evolve naturally with the interaction of the system under consideration. But, since only a fraction of the computer system is observable, these models are usually inaccurate and incomplete.

In computer systems, the formal counterpart of a mental model is often called an ontology. An ontology explicitly states the conceptualisation of a particular domain in a formal language. It abstracts the essence of the domain of interest and helps to catalogue and distinguish various types of objects in the domain, their properties and relationships (see, e.g. [6]). An ontology also enables the system to reason about various aspects of the domain.

Since the mental model of the application domain is usually partial, inconsistent and subject to change, conceptual discrepancies with the system's ontology are more the rule than the exception. Clearly, detection of these flaws is of crucial importance. But how does a system become aware that the user's mental model deviates from the system's ontology? One possibility is that the system observes that the user applies an incorrect action to a particular object. Another is that the user's dialogue contribution contains the wrong words, words in the wrong order, or an incorrect combination of words. To illustrate the latter: suppose a user requests the system to 'edit process 308', the system may conclude that, in contrast to the system's belief, the user believes that processes can be edited.

Below we will concentrate on detection of discrepancies on the basis of the user's communicative behaviour. We will assume that the system is able to interpret a simple language fragment. It detects discrepancies on the basis of particular inferences drawn from the message. These inferences will be called 'presuppositions', a kind of background assumptions that should be fulfilled in order to understand the meaning of the message. Presuppositions will be compared with the system's ontology and feedback will be generated on the basis of an incorrect match between the two types of information.

3. FEEDBACK

In both human-system and human-human communication, feedback is used for a broad range of communicative responses at various levels and has an enormous diversity, varying from a simple nod or a beep that indicates the receipt of a message to a written comment that evaluates the quality of a scientific paper. However, for various reasons, we have no accurate mathematical theory for natural communicative behavior and the application of cybernetic models to human communicative activities has only a limited scope of relevance [7].

In human-system interaction - where a system is represented by some kind of electronic equipment, such as a computer or a video player - a diversity of heuristics for feedback is suggested. Nielsen, for instance, states that a system should continuously inform the user about what it is doing and how it is interpreting the user's input [3]. More detailed heuristics concern the different degrees of persistence in the interface, respons times and corrective feedback in case of errors.

When we look at feedback phenomena in conversations between humans, sequences in terms of speech acts appear to be rather chaotic and seem hardly subjected to any rules. Questions can be followed by answers, denials of the relevance of the question, rejections of the presuppositions of the question, statements of ignorance, and so on. An example of general rules for cooperative contributions, and conversational feedback in particular, are the Gricean maxims for conversation, such as 'tell the truth' (quality), 'say enough, but not too much' (quantity), 'be relevant' (relevance) and 'use the appropriate form' (manner) [2]. Clearly, these rules are still vague and not all people follow them to the letter, but Grice's point is that, contrary to particular appearances in conversation, the principles are still adhered to at some deeper level.

Just as the Gricean maxims form guidelines for the acceptability of human conversational sequences, Nielsen's heuristics offer an important and practical handle for a systematic evaluation of user interfaces. However, both type of rules are underspecified in case an interface designer wants to realize the actual implementation. In other words, the rules have some explanatory power, but no predictive power and do not provide the designer with sufficient detail about the type, content and form of the feedback that has to be generated in a particular situation. Suppose, for instance, that user U and computer system S have two disparate conceptualizations and that U asks the question: 'Is this file running?'. S's ontology contains, among other things, a representation for the words 'file' and 'running' (and knows which file is referred to) and knows that files are a subclass of items. S also knows that running is only applicable to processes and that processes are not items. Assuming that our computer system should be relevant and truthful, then what should the response of S be? Clearly, we have abundant possibilities for feedback:

S: 'Sorry, I don't understand you.'
S: 'What do you mean by 'running'?'
S: 'Running is only applicable to processes.'
S: 'Running is not applicable to files.'
S: 'Files are not processes.'
S: 'Items are not processes.'
S: 'Do you think that running is applicable to files?'
S: ...

Which utterance is the most adequate one and which rules should we apply in order to generate these feedback sequences? This depends, for instance, on what the user and system know about the application domain and, more specifically, on the system's knowlegde about the user's conceptualization. Another parameter is the role played by the system in the interaction; usually, the system acts as an expert who is unwilling to adjust its own ontology, in other cases the system may react as an equal who is willing to consider its domain ontology.

4. ONTOLOGIES IN TYPE THEORY

In this paper, the system's ontology is expressed in type theory (TT). TT, which is actually based on typed lambda calculus, is a powerful logical formalism in the field of theorem proving and programming languages. In the field of human-computer communication, TT was used in the DenK-project [1] as a knowledge representation of the system to model various types of beliefs. In the project, an 'intelligent' agent was modelled that supported a human user in its use of a particular domain. Although the system was applied to the domain of an electron microscope, it was intended to be generic in that its architecture and various techniques for knowledge representation were independent of the field of application. The agent's belief states, such as private and common beliefs, were modelled as type theoretical contexts. In the DenK-project the formalism was used for modelling the ontological assumptions and beliefs about the task domain (the electron microscope) and the cognitive dynamics of the agent's belief state, in particular, the change of beliefs as a result of domain observations and dialogue contributions by the user.

Here we will only give a brief and informal introduction to TT and show how particular ontological information can be expressed in a very limited fragment. Beliefs of an agent can be represented in TT as so called contexts; contexts consist of sequences of expressions and list everything that has been assumed so far and everything that has explicitly been inferred from these assumptions. The building blocks in TT are expressions of the form

$$G : H \tag{1}$$

which indicate that an object G has type H. These expressions are called statements. We can express, for instance, that a particular object (namely p_{308}) is an 'inhabitant' of type *process*:

$$p_{308} : process \tag{2}$$

Concepts themselves are inhabitants of a special type named 'sort' (which is denoted by $\star_s$). For instance, the statement

$$process : \star_s \tag{3}$$

expresses that *process* is a concept.

The notion of sequentiality plays a role in the order in which statements can be added. For instance, (2) cannot be added unless the belief state of the agent already contains statement (3).

Predicates are considered as a functions that take a particular concept as an argument and that yield a truth value. For that reason, we also have to introduce propositions; we will do this by introducing a new kind of sort '$\star_p$' For instance, the statement

$$running : process \to \star_p \tag{4}$$

expresses that the predicate running takes inhabitants of *process* as an argument and yields propositions as its result.

Inheritance is introduced by the subsumption operator '$<$', which indicates that inhabitants of a more specific type can be applied in every case where inhabitants of the more general type may be applied (types on the left are lower in the hierarchy):

$$item : \star_s \tag{5}$$

$$file < item : \star_s \tag{6}$$

$$open : item \to \star_p \tag{7}$$

So, the predicate *open* takes as an argument inhabitants of the type *file* as well.

Contexts are open to new information and can be extended as long as new introductions are adhered to the rules of the type system; these contexts are called *legal contexts*. So, given a particular context, the rules of the type system constrain the way in which statements can be combined into new legal statements. This can be expressed by so called judgements:

$$\Gamma \vdash E : T \tag{8}$$

which expresses that term E has type T, given the assumptions in context Γ. The rules of the type system may guarantee, for instance, monotonicity and inheritance. For instance, given the statement

$$f_{202} : file \tag{9}$$

we can derive from the context Γ consisting of the statements (5), (6), (7) and (9) the following judgement:

$$\Gamma \vdash open(f_{202}) : \star_p \tag{10}$$

Hence, given our previous introductions, if we apply the predicate *open* to f_{202}, we can legally extend the context by the new statement implying that $open(f_{202}) : \star_p$ is a legal extension of the existing context.

Presuppositions will be expressed as lists of TT-statements. We will say that a list of presuppositions is a legal extension of a particular context if this context extended with the presupposition list is still a legal context.

5. RULES FOR DETECTION AND FEEDBACK

We consider incoming information of the form:

$$\text{Is this } w_1\ w_2\text{ ?} \tag{11}$$

$$\text{Is this } w_1 \text{ a } w_2\text{ ?} \tag{12}$$

The former message denotes a query whether a particular object satisfies a particular predicate while the latter denotes a query whether a particular object is an instance of a particular category (we will also allow 'an' instead of 'a').

A communication language has semantic and pragmatic aspects. The semantics deals with how words (i.e. w_1 and w_2) are interpreted in terms of the underlying ontology. The pragmatics deals with the aspects that are independent of the ontology; e.g., the type of the message (e.g. query or assertion), the definite article 'this' and the indefinite article 'a' and so on.

The first stage of our method concerns the extraction of the presuppositions that are inherent in the message. This is done on the basis of the system's interpretation of the words in terms of its ontology (modelled by an interpretation function I mapping words to the ontology) together with the pragmatics of the communication language. Query (11) presupposes the existence of a concept $I(w_1)$, a particular object that is an inhabitant of this concept (we assume that the computer knows which object is under discussion) and a predicate $I(w_2)$ that is applicable to the concept $I(w_1)$. Formally:

$$I(w_1) : \star_s,\ \ x_{17} : I(w_1),\ \ I(w_2) : I(w_1) \to \star_p \tag{13}$$

Query (12) presupposes the existence of a concept $I(w_1)$, a particular object that is an inhabitant of this concept and the existence of a concept $I(w_2)$ that is a subtype of $I(w_1)$. Formally:

$$I(w_1) : \star_s,\ \ x_{17} : I(w_1),\ \ I(w_2) < I(w_1) : \star_s \tag{14}$$

For instance, let the system's ontology O be defined as:

$$\begin{array}{l} item : \star_s \\ executable : \star_s \\ open : item \to \star_p \\ x_{267} : item \\ y_{149} : executable \end{array}$$

Figure 1 depicts the four major relationships that can hold between the ontology O and the presuppositions P. In case (a), the presuppositions are part of the ontology (denoted by $P \subset O$). An example of this is the question:

$$\text{Is this item open?} \tag{15}$$

whose presuppositions are (using (13)):

$$\begin{array}{l} item : \star_s \\ x_{267} : item \\ open : item \to \star_p \end{array}$$

(We assume that the interpretation function makes the obvious mapping from the words of the communication language to the ontology.)

In case (b), the presuppositions constitute a legal extension of the ontology. An example of this is the question:

$$\text{Is this item an executable?} \tag{16}$$

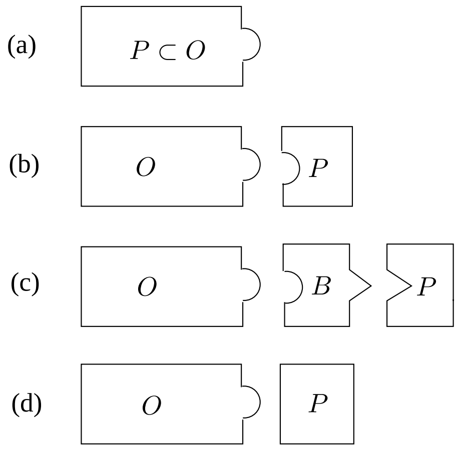

Figure 1: Four different relationships between ontology O and presuppositions P.

whose presuppositions are (using (14)):

$$\begin{aligned} &item : \star_s \\ &x_{267} : item \\ &executable < item : \star_s \end{aligned}$$

According to the rules of the type system the statement $executable < item : \star_s$ constitutes a legal extension of the ontology O.

In cases (c) and (d), the presuppositions do not constitute a legal extension of the ontology. In case (c), there exists a type theoretical context B that bridges the gap between the presuppositions and the ontology (cf. [4]). B together with the presuppositions constitutes a legal extension of the ontology. An example of this is the question:

Is this executable open? (17)

whose presuppositions are:

$$\begin{aligned} &executable : \star_s \\ &y_{149} : executable \\ &open : executable \rightarrow \star_p \end{aligned}$$

These presuppositions do not constitute a legal extension of the computer's ontology O. According to the rules of the type system, it is only possible for a predicate to be applicable to different types if one of these is a subtype of the other. The gap can be bridged by the statement:

$$executable < item : \star_s$$

In other words, if the system adds the information that executables are a subclass of items, the discrepancy can be resolved. Note that in this case, the system throws off its expert role.

Also, in case (d), there is a gap, but there is no bridging context. An example of this is the question:

Is this item executable? (18)

whose presuppositions are:

$$\begin{aligned} &item : \star_s \\ &x_{267} : item \\ &executable : item \rightarrow \star_p \end{aligned}$$

Since, according to the rules of the type system 'executable' cannot be both a concept (i.e. of type $\star_s$) and a predicate (i.e. of type $item \rightarrow \star_p$), the presuppositions are incompatible with the ontology.

What ontological feedback should be generated in each of the four cases? In case (a), there are no discrepancies. The rule is to answer the question (for instance, in accordance with the Gricean maxims of cooperation).

In case (b), the rule is to report that the derived presuppositions were not part of the system's ontology. So the response to (16) could be:

I did not know that executables are items. (19)

In case (c), the rule is to ask whether a particular bridge is part of the user's mental model. So, the response to (17) could be:

Do you think that executables are items? (20)

In case (d), a conflict has been detected. The rule here is to state the discrepancy. So, the response to (18) could be:

Executable is a concept and not a predicate over items. (21)

6. CONCLUSIONS

In this paper, a rudimentary framework was presented to generate feedback utterances in a dialogue between a computer system and its user. For that, we included an explicit representation of the application domain, a so-called ontology, and a method for detecting conceptual discrepancies between the system's ontology and the user's mental model.

In the future, we will extend the basic framework to richer ontologies and we will give concrete form to the feedback rules based on, for instance, various types of belief about the user and the type of discrepancy. Another important aspect is the acquisition of empirical data to determine what humans actually do in realistic conversational circumstances. Although we have taken a strong theoretical stance, we believe that the various aspects described in this paper form the minimal ingredients that enable a system to generate adequate feedback utterances at the conceptual level in human-computer interaction.

7. REFERENCES

[1] R.M.C. Ahn, R.J. Beun, T. Borghuis, H.C. Bunt, and C.W.A.M. van Overveld. The DenK-architecture: A fundamental approach to user-interfaces. *Artificial Intelligence Review*, 8:431–445, 1995.

[2] H.P. Grice. Logic and conversation. In P. Cole and J.L. Morgan, editors, *Speech Acts. Syntax and Semantics, Vol. 11*, pages 41–58. Academic Press, New York, 1975.

[3] J. Nielsen. *Usability Engineering*. Morgan Kaufmann, San Diego, CA, 1993.

[4] P. Piwek and E. Krahmer. Presuppositions in context: Constructing bridges. In P. Bonzon, M. Cavalcanti, and R. Nossum, editors, *Formal Aspects of Context*, volume 20 of *Applied Logic Series*. Kluwer Academic Publishers, 2000.

[5] D. Redmond-Pyle and A. Moore. *Graphical User Interface Design and Evaluation*. Pearson Education, Harlow, UK, 1995.

[6] J.F. Sowa. Top-level ontological categories. *Int. J. Hum.-Comput. Stud.*, 43(5-6):669–685, 1995.

[7] A. Spink and T Saracevic. Human-computer interaction in information retrieval: Nature and manifestation of feedback. *Interacting with Computers*, 10:249–267, 1998.