Multimodal Cooperation with the DenK System

Harry Bunt[1], René Ahn[1], Robbert-Jan Beun[2], Tijn Borghuis[3], and Kees van Overveld[3]

[1] Computational Linguistics and Artificial Intelligence Group, Tilburg University, P.O. Box 90153, 5000 LE Tilburg, The Netherlands, email: {bunt,ahn}@kub.nl)

[2] Center for Research on User-System Interaction (IPO) P.O. Box 513, 5600 MB Eindhoven, The Netherlands email: rjbeun@ipo.tue.nl

[3] Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands email: {tijn, wsinkvo}@win.tue.nl

Abstract. In this chapter we present the DenK project, a long-term effort where the aim is to build a generic cooperative human-computer interface combining multiple input and output modalities. We discuss the view on human-computer interaction that underlies the project and the emerging DenK system. The project integrates results from fundamental research in knowledge representation, communication, natural language semantics and pragmatics, and object-oriented animation. Central stage in the project is occupied by the design of a cooperative and knowledgeable electronic assistant that communicates in natural language and that has internal access to an application domain which is presented visually to the user. This electronic *'Cooperative Assistant'* has an information state that is represented in a rich form of type theory, a formalism that enables us to model the inherent cognitive dynamics of a dialogue participant. This formalism is used both for modeling domain knowledge, for representing the current dialogue context, and for implementing a context-change theory of communication.

1 Introduction

The DenK project is long-term collaborative research activity of the universities of Tilburg and Eindhoven that aims at the exploration, formalization and application of fundamental principles of communication from a computational perspective, in order to build advanced cooperative human-computer interfaces.[4] The project combines fundamental research in knowledge representation, communication, natural language semantics and pragmatics, and object-oriented animation.

Point of departure in the DenK project is that, from a user's point of view, a computer should ideally present itself as an intelligent 'electronic assistant' who

[4] 'DenK' is an abbreviation of 'Dialoogvoering en Kennisopbouw', which means 'Dialogue Management and Knowledge Acquisition'. The word *denk* in Dutch means *think*.

is knowledgeable about the task and the domain of the application and interacts in an intelligent and cooperative way with the user, using natural language and other modalities as appropriate. The DenK system, that should instantiate this ideal, is intended to be *generic* in that its architecture, as well as the techniques developed and incorporated in the various modules and interfaces, should be applicable over a wide range of application domains and tasks.

In this chapter we first discuss the motivation and the fundamental principles that underlie the architecture and functionality of the DenK system. In subsequent sections we discuss the research concerned with knowledge representation and reasoning, natural language interpretation, pragmatics and dialogue management, and visual domain representation. We describe the design of the major components in a first, preliminary prototype of the system to be developed. In conclusion, we consider some of the main results obtained thus far, and outline the research agenda for the near future.

2 Human-Computer Interaction and Communication

The DenK project takes an approach to communication according to which communicative behaviour is analysed in terms of actions, motivated by a goal, purpose, or function. This is the general idea of speech act theory (Austin, 1962; Searle, 1969) as well as of more recent approaches to dialogue analysis such as *Dynamic Interpretation Theory* (Bunt, 1991; 1994; 1997) and *Communicative Activity Analysis* (Allwood, 1976; 1997). The motivation for communicating may in the case of human-human interaction be either of a social character, such as being friendly, seeking support, or courting; or it may concern a specific task such as solving a planning problem. In the case of human-computer communication, social motivations do not arise, so we may assume that the user communicates with a computer in order to accomplish a certain task - what is often called an *application*. Many kinds of application, for instance in areas such as process control and computer-aided design, involve real-world objects or models of such objects. The interaction with such objects is physical, in the form of such actions as picking up, turning around, or fastening, and perceptual, in terms of seeing, feeling, hearing, or smelling. Interaction with other *agents*, by contrast, is in symbolic form, using signalling systems such as natural language, gestures, and facial expressions. Symbolic actions, like speech acts, require an interpreter to bridge the gap between symbols and their actual meaning and purpose, before they can produce changes in the domain of physical objects, or can provide information about these objects. Physical acts, by contrast, directly change the application domain, while perceptual acts directly provide information about the domain (see also Hutchins, 1989; de Souza, 1993)

The distinction of two types of interaction, direct and indirect, is clearly reflected in the design of the DenK system, with two components playing distinct roles in the user-system interaction:

a. the *Cooperative Assistant*, which supports symbolic interaction; it interprets messages in natural language from the user, is capable of reasoning about various aspects of the domain and the user, and produces communicative behaviour adequate with respect to (its model of) the user's beliefs and goals;

b. the *Application Domain Model*, implemented by means of an animation system which incorporates spatio-temporal components and graphical tools for the representation and visualization of the domain. The user can visually observe the domain, in particular see the events that take place, and can operate directly on domain objects with graphical actions.

The user of the DenK system can thus interact with the application domain both (not: *either*) indirectly, through linguistic communication with the Cooperative Assistant who has internal access to the domain, and (not: *or*) directly, through graphical operations. We emphasize *both .. and* instead of *either .. or*, since the user can also interact by using these two modalities in combination, for instance by using a deictic expression and grasping an object on the screen. This is depicted in the *'triangle view'* of interaction, depicted in Fig. 1, where we have three interacting components: the User, the Cooperative Assistant, and the Domain Model. We will see below that the overall design of the DenK system is based on this triangle view.

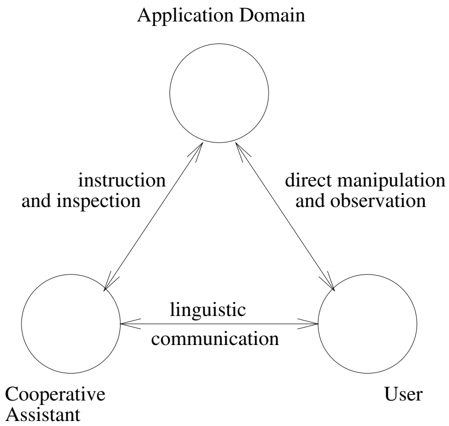

Fig. 1. The DenK *'triangle view'* of interaction: User, Cooperative Assistant, and Application Domain.

It may be noted that the overall architecture of the DenK system, in reflecting the triangle view, differs from that of other intelligent multimedia systems, such as AIMI (Burger & Marshall, 1993), WIP (Wahlster et al., 1993), MMI^2 (Wilson et al., 1991) and CUBRICON (Neal & Shapiro, 1991), in which there is no direct link between the user and the application domain. An advantage of our approach is that certain particularly complex aspects of the interaction, such as the visualization of autonomous motion behaviour of objects, do not have to be considered by the Cooperative Assistant and can be left to the interactive component of the application domain model.

In order to be able to act as the ideal electronic cooperative assistant, the DenK system's Cooperative Assistant should have the properties required of a communicative agent, as laid out in (Bunt, 1997b):

1. Be a knowledgeable communicative agent, i.e., be competent in the use of the available modalities and their combinations. This requirement includes knowing what types of communicative actions van be performed, what action is appropriate to perform under what conditions, and what is an appropriate form of an action in a given context.
2. Be informed about the domain of discourse and the current dialogue context. Knowledge of the domain of discourse is indispensable for understanding the user, since understanding involves relating the elements of an input message to the semantic concepts that make up their meanings. Knowledge of the current dialogue context can be divided into the following categories (Bunt, 1997a):
   (a) *linguistic context*, most importantly the recent dialogue history;
   (b) *semantic context*, i.e. the current state of the domain of discourse;
   (c) *cognitive context*, including the current beliefs about the user, and information about the current status of cognitive processes (such as the information that the interpretation of the last input message needs additional information to resolve an ambiguity);
   (d) *physical and perceptual*, relating to the physical conditions of the interaction and the perceptual availability of information;
   (e) *social*, describing current 'pressures' to perform particular communicative actions (such as making an apology, or providing feedback on the result of carrying out an instruction).[5]
3. Have reasoning capacities. To apply his knowledge of communication, of the application domain, and of the current dialogue context, an agent has to be able to combine pieces of information in logically correct ways. Understanding and evaluating the content and function of messages involves making such combinations, and thus involves reasoning, as do the planning and generation of noncommunicative as well as communicative acts.
4. Be rational. Being able to reason is not sufficient. The ability to draw correct inferences should be used in a rational way, to compute for instance the intentions and other relevant conditions of (communicative or other) actions that are instrumental to achieving a certain goal or purpose, or to infer subgoals that are on the way to a higher goal. Rationality directs the use of reasoning power.

[5] Social aspects of context do not only play a role in human-human communication, but also in human-computer interaction. Acknowledgements and apologies, for instance, help to make clear to the user to what extent the communication is proceeding successfully.

5. Be social. Knowing the rules and conventions of using a communicative system (in a certain type of interactive situation, within a community) is not sufficient: one must also have the *disposition* to act accordingly. An important aspect of this is that a social agents takes other participants' goals, desires, beliefs, and other properties into account, in particular so as to help others to achieve their goals - which is commonly called *being cooperative.*

In projects that aim at building multimodal communication systems, there is a tendency to start working on the processing of inputs from several modalities, or on the use of modalities for output presentation; this may be explained by the fact that the forms of inputs and outputs are conspicuous aspects of multimodal systems. As a consequence, many of the requirements just listed tend to receive relatively little attention. In the DenK project, we decided to work the other way round, in an 'inside-out' fashion, developing first (a) a powerful representation of dialogue contexts, and (b) a model of the mechanisms for context change under the influence of interaction with the user, as well as (c) a model of the way changing contexts give rise to communicative activity. This approach can be viewed as implementing the context-change view of communication in a system-internal knowledge representation formalism, using an architecture that allows multiple input and output modalities for the expression of communicative action, and adding a number of modalities in a later stage.

In the presentation in this chapter we follow the same approach, considering first to the use of the powerful representation formalism that was chosen and further developed in the DenK project for context representation and formal communication modeling.

3 The DenK System

As mentioned above, the DenK system is intended to be *generic* in that its architecture, as well as many of the techniques developed and incorporated in the various modules and interfaces, should be applicable over a wide range of application domains and tasks. To demonstrate the generic character of the system, several applications are envisaged, the most important one being a training simulator for the use of a modern electron microscope as developed by PHILIPS Electron Optics. This is an important instrument for materials research in physics, as well as for medical research in pathology. The device is very complex, and uninitiated users have to go through an intensive training period in order to learn how to use it. This is primarily due to the difficulty that users have in forming an adequate picture of the internal workings of the device (the relevant parts, their functions, and their relations), that manifests itself to users essentially as a black box. Offering trainees the possibility to exercise with an interactive training simulator is expected to considerably shorten the training period. We will refer to this application of the DenK system under construction as the *'DK-2 system'* (see Bunt et al., 1998).

For experimental purposes, a preliminary partial prototype of the DenK system has been implemented, from now on referred to as the *'DK-1 System'*, where the application domain is a blocks world, with toy blocks of different shapes, sizes and colours, with specifiable autonomous movements in 3D space such as rotating, moving to or from a distance, etc. The blocks world has a 3D representation on the screen by means of an animation system described below. This DK-1 prototype does not allow direct manipulation of objects by the user, and supports only simple dialogue behaviour and a toy fragment of natural language. The DK-1 system can execute simple commands to act on the domain and answer questions about its current state.

A simple way in which the current system uses context information is that, when answering questions, it takes the preceding dialogue into account and supplies only information which, according to its context information, is not shared with the user. The system also uses context information (viz. domain knowledge) to detect presupposition violations. If the user gives a command that is impossible to perform, the system will detect and report this.

The project's 'inside-out' strategy, concentrating first on developing the formalisms and techniques for context representation, reasoning and domain modeling, has resulted in the availability in the DK-1 prototype of the Application Domain Model component, and in the representation-, reasoning-, evaluation-, and visualization systems that form the backbone of the system.

4 Knowledge Representation and Reasoning

4.1 Type Theory

For modeling the information state of the Cooperative Assistant and the way it changes under the influence of the communication, we use Constructive Type Theory (CTT), a versatile and powerful logical formalism for knowledge representation and reasoning, based on typed lambda calculus. The language of (explicitly) typed lambda calculus consists of statements of the general form $\mathsf{V} : \mathsf{T}$, expressing that an object V has type T. Type-theoretical statements '$\mathsf{V} : \mathsf{T}$' should be read: 'V is an object of type T', or 'V is an inhabitant of T'. Such expressions are called 'entries' or 'introductions'. The subexpressions 'V' at the left-hand side of an introduction are variables, those at the right-hand side are types. Entries can be grouped into sequences, which in type theory are called *contexts*.

Types are atomic or complex. Atomic types come in two varieties: constants and variables. A variable x may be used as a type, provided that it has been introduced in the context in which occurs (i.e., there is an introduction of the form $\mathsf{V} : \mathsf{T}$ earlier in the sequence that forms the current context, where an expression of the form $\mathsf{A} : \mathsf{V}$ occurs). Constant atomic types can be used at any point in a context, and are also called *'sorts'*. Among the sorts is prop, for propositions, and class, for objects of any other kind. Two complex types of importance here are function types, formed by means of '*Π-abstraction*' (also called '*Π-formation*'), and the types formed by function application.

Function types are defined as follows. If t_i and t_j are types and x is a variable then $\Pi\mathsf{x}{:}\mathsf{t}_i.\mathsf{t}_j(\mathsf{x})$ is a function type. If f is a function of this type, and y is an inhabitant of type t_i, then $\mathsf{f(y)}$ is an inhabitant of the type $\mathsf{t}_j(y)$. A special case is that where the variable x does not occur in $\mathsf{t}_j(\mathsf{y})$, in that case we use the simpler notation $\mathsf{t}_i \rightarrow \mathsf{t}_j$, instead of $\Pi\mathsf{x}{:}\mathsf{t}_i.\mathsf{t}_j(\mathsf{x})$.

A *context* can now be defined more precisely as a well-formed sequence of introductions, well-formed in that every type occurring in the right-hand side of an introduction is either a sort or is the left-hand side of an introduction earlier in the sequence. A variable functions as a constant in the part of a context that follows its introduction.

Whether a type-theoretical statement $\mathsf{V} : \mathsf{T}$ is true has to be decided on the basis of the context in which it occurs: the preceding sequence of statements should either already contain that expression, or it should be possible to obtain the expression from the preceding context by means of the type inference rules of the logic.

Typed lambda calculi have been applied in foundational mathematical research, which was their original motivation, as well as in the mathematical study of programming languages and the semantics of natural language. Barendregt (1991) noticed that many existing systems of typed lambda calculus can be uniformly represented using the format of *Pure Type Systems* (see also Barendregt, 1992). Formalisms within this class include *Automath* (de Bruijn, 1980) and the *Calculus of Constructions* (Coquand, 1985); closely related is also *intuitionistic type theory* (Martin-Löf, 1984). The CTT formalism, used in the DenK project, is a variant of Pure Type Systems with extensions relating to the needs of dialogue context modeling.

A crucial and peculiar point of type theory is that *propositions can be treated as types*, and inhabitants of a proposition as proofs of that proposition; a proposition itself has type 'prop'. Propositions thus occur in type-theoretical expressions in the same way as individual concepts, and proofs occur as individual objects. Proofs and other objects may be combined to form new proofs and other objects. The inference rules of the type system restrict the way in which this can be done, and guarantee that all reasoning is sound.[6] Type theory thus takes proofs to be first-class citizens: proofs are considered as abstract objects, have representations in the language, and are fully integrated within the formalism. This means that in type theory we can represent not only *what* an agent believes, but also *how* he comes to believe it, by having explicit representations of the proofs that justify his beliefs.

Type theory is primarily concerned with the recording of information and with what can be effectively shown to follow from it. An agent whose beliefs are represented type-theoretically, is regarded as *explicitly* believing those propositions that are present in the type-theoretical context, and *implicitly* believing those propositions that are not explicitly present but for which he can effectively construct a proof, given the available axioms, deduction rules and proof-construction strategies. (Once the agent has constructed such a proof, he can record it in his context, the belief then becomes explicit.) The framework is thus

[6] This ingenious idea is known as the *Curry-Howard-De Bruijn isomorphism* (Curry & Feys, 1958).

constructivist in the sense that effective constructability of a proof is central, rather than truth.

The total set of the agent's beliefs is thus determined by what the agent is able to deduce from the type-theoretical context. For some propositions `p`, the agent will not be able to construct a proof, nor will he be able to construct a proof of `not p`, hence this framework constitutes an inherently *partial* approach to belief modelling.

In recent years, research in the DENK-project has shown the great potential of type-theoretical contexts as formalizations of context information, provided the expressive capabilities and proof methods of standard type theory are extended in order to model states of information and intention of agents participating in a dialogue (Borghuis, 1994; Ahn, 1995a; Jaspars, 1993; Beun & Kievit, 1996). An extension with epistemic modalities has been defined by Borghuis (1994), which allows the formal modelling of beliefs with different degrees of certainty, and of distinguishing 'private' beliefs from assumed shared beliefs. Further extensions are required for the type-theoretical representation of the time-dependent aspects of the behaviour of objects in the application domain, and also for the representation of and the reasoning about the temporal aspects of natural language utterances (see Ahn, 1996; Ahn & Borghuis, 1996).

In the DENK-project, CTT is used for implementing the semantic representation of the utterances exchanged by the user and the system; for implementing representations of the system's knowledge of the task domain (the global semantic context, which is assumed to be shared with the user); and for representing the shared beliefs of user and system derived from the dialogue, (i.e. the system's view of the 'local semantic context'). The representation of local semantic context is implemented in a simple way by dividing a CTT context into two parts, `'Common'` and `'Private'`. `Common` is the part where the system stores the information it believes to be shared with the user; `Private` contains the beliefs the system does not believe to be shared with the user. The two subcontexts are taken as primitives, which means that modalities are not modelled in the CTT language.[7]

In the following section we show, by means of a simple example, how the information state of the Cooperative Assistant can be represented by a type-theoretical context.

4.2 Information States as Type-theoretical Contexts

We consider the blocks world domain of the DK-1 prototype system, where we have pyramids, cubes and the like, which can move around and have properties like colour and size. In this example we assume the Cooperative Assistant to have the following information:

[7] Alternatively, epistemic operators can be defined in the object language of CTT; Borghuis (1994) shows how modalities can be modelled explicitly by extending CTT with deduction rules for entering and exiting subcontexts. The implementation with contexts separated into parts is computationally simpler, however, and adequate for the purposes of the DENK-project.

– there may be pyramids and colours in the domain (the Cooperative Assistant is familiar with the notions 'pyramid' and 'colour'; see below);
– 'small' is a predicate applicable to pyramids, and 'bright' is a predicate applicable to colours;
– there is actually a pyramid, which is small;
– all pyramids have a colour, and all pyramids have a bright colour.

To describe the Cooperative Assistant's information state, we begin with an empty context and gradually introduce the information listed above. First, we have to introduce the necessary types corresponding to the concepts we assumed, i.e. 'pyramid' and 'colour'. In type theory, new types may be introduced as inhabitants of one of the sorts, or of one of the types already introduced. So we construct the following context:[8]

$$[\ \mathsf{pyramid} : \mathsf{class}\ ,\ \mathsf{colour} : \mathsf{class}\ ]$$

Once a type has been introduced, inhabitants of that type can be introduced. The knowledge that there actually is a pyramid in the (current state of the) domain is expressed in CTT by introducing an individual of that type, so we extend the context with the entry:

$$\mathsf{p}_{38} : \mathsf{pyramid}$$

(using the variable 'p_{38}' as an arbitrary name for this individual).

Predicates are represented in CTT as functions to propositions, so the predicate 'bright' is a function that, applied to a colour, yields a proposition like: 'Yellow is bright'. Propositions are treated as types, as mentioned above, and occur in CTT expressions in the same way as the types 'pyramid' and 'colour' in the present example. We thus introduce the predicate 'small' as a function from pyramids to propositions, and the predicate 'bright' as a function from colours to propositions (objects of type 'prop'). These are added to the example context by extending it with the entries:

$$\mathsf{small} : \mathsf{pyramid} \to \mathsf{prop}$$
$$\mathsf{bright} : \mathsf{colour} \to \mathsf{prop}$$

The knowledge that the particular pyramid 'p_{38}' is small is expressed type-theoretically by saying that there is a proof of this fact. This corresponds to adding the entry:

$$\mathsf{pr}_1 : \mathsf{small}(\mathsf{p}_{38})$$

(where 'pr_1' is an (arbitrary) name of the proof). We can now express the Cooperative Assistant's knowledge that pyramids have a colour, and that all pyramids

[8] The entries making up a context are separated by commas; beginning and end of a context are marked by '[' and ']', respectively.

have bright colours. We first extend the context with an entry representing a function, 'col', that associates colours to pyramids:

$$\mathsf{col} : \mathsf{pyramid} \to \mathsf{colour}$$

To represent that the colour of a pyramid is always bright type-theoretically, we have to express that for every pyramid we have a proof that its colour is bright. To this end we introduce a function f that, given a pyramid x, returns a proof that the colour of x is bright. In other words, $\mathsf{f(x)}$ is an object of type $\mathsf{bright(col(x))}$. The following introduction achieves this:

$$\mathsf{f} : \Pi \mathsf{x{:}pyramid.bright(col(x))}$$

Combining all the above entries results in the following context, which represents the beliefs of the Cooperative Assistant about the domain:

$$\begin{array}{l} [\ \mathsf{pyramid} : \mathsf{entity}\ ,\ \mathsf{colour} : \mathsf{entity}\ , \\ \ \ \mathsf{p}_{38} : \mathsf{pyramid}\ , \\ \ \ \mathsf{small} : \mathsf{pyramid} \to \mathsf{prop},\ \mathsf{bright} : \mathsf{colour} \to \mathsf{prop}\ , \\ \ \ \mathsf{pr}_1 : \mathsf{small}(\mathsf{p}_{38})\ , \\ \ \ \mathsf{col} : \mathsf{pyramid} \to \mathsf{colour}\ , \\ \ \ \mathsf{f} : \Pi \mathsf{x{:}pyramid.bright(col(x))}\ ] \end{array}$$

Once the information is represented in the form of a type-theoretical context, the Cooperative Assistant can make inferences by constructing new objects using the entries in the context. In fact, this representation can be used to support an effective proof construction method for type theory, combining resolution style proofs with natural deduction (Helmink & Ahn, 1991). The Cooperative Assistant incorporates a theorem prover based on this method.

4.3 Communication in Type Theory

The Cooperative Assistant is responsible for the communication with the user, i.e. it has to interpret the user's utterances and to generate appropriate responses. We mentioned above that the DenK system is based on a view of communication as change of local context, where *local context* is viewed, following Dynamic Interpretation Theory, as the totality of information that may change through communication (Bunt, 1994). The full context that is relevant for the interpretation and generation of communicative behaviour has, besides this local part, also a *global part*, consisting of information that is not changed through communication. The DenK system, or rather the Cooperative Assistant, is viewed as an expert w.r.t. the application, so its beliefs about the application cannot be changed by the user; it forms part of the global context. The only possible change in application-related information is that the system comes to know that the user shares some belief with the system. This is then represented in `Common`, the CTT-subcontext containing the 'shared beliefs', In the current design of the DENK system, we therefore restrict the notion of 'local context' to the shared beliefs about the application domain, and represent this as a type-theoretical context. (Other kinds of local context information, such as information about the status of perceptual, interpretive and inferential processes, could in principle also be represented in type-theoretical form; see Bunt, 1997b.)

In order to interpret the user's utterances as context-changing actions, the Cooperative Assistant incorporates components for analysing utterances and representing their meanings in CTT. These are discussed in Sections 5 and 7. In the present section we consider the user-system communication at the formal level of type theory, i.e. for the moment we take the conversions between linguistic (and gestural) form and type theory for granted.

We consider three cases in which the Cooperative Assistant generates a pragmatically correct reply to an utterance of the user, using contextual information. We assume the information state of the Cooperative Assistant to be the one represented above, and we assume that all beliefs of the Cooperative Assistant are shared with the user except for the constraint that every pyramid has a bright colour, which is a 'private' belief of the Cooperative Assistant. Note that the Cooperative Assistant is assumed to be an expert about the domain and that all declarative utterances about the domain contributed by the user are therefore interpreted not as statements but as verification questions.

Having inspected the visual representation of the state of the application domain on the screen, the user might produce the utterance:

"The pyramid is small."

In order to interpret this utterance, the Cooperative Assistant has to figure out which pyramid is meant. In view of the definiteness of the noun phrase, the Cooperative Assistant may consult both the application domain and the shared beliefs. The Assistant's private beliefs are irrelevant here, because the user can have no knowledge of these. The Cooperative Assistant will therefore assume that the user refers to the pyramid about which a shared belief is stored in his `Common` subcontext ('$\mathsf{p}_{38} : \mathsf{pyramid}$'). He will interpret the user's utterance as saying that there is evidence that $\mathsf{small}(\mathsf{p}_{38})$ holds. After checking the truth of this proposition in the application domain, the Cooperative Assistant extends his `Common` subcontext with the following entry:

$\mathsf{e}_5 : \mathsf{small}(\mathsf{p}_{38})$

where e_5 labels the evidence the Cooperative Assistant has found for the proposition $\mathsf{small}(\mathsf{p}_{38})$. The Cooperative Assistant will give the answer *"yes"*, to indicate that he agrees. The proposition $\mathsf{the\ pyramid\ is\ small}$ becomes a shared belief of Cooperative Assistant and user. If, on the other hand, the Assistant can prove the proposition to be false, a *"no"* answer will be generated; if the Assistant cannot establish the truth of the proposition, for instance because no referent can be found for *"The pyramid"*, then this will be reported to the user.

Suppose that the next utterance of the user is:

"Is the colour of the pyramid bright?"

The communicative function of the utterance is a YES/NO-QUESTION; the user signals that he wants to know whether the proposition the pyramid is bright holds. The Cooperative Assistant complies with this wish by trying to construct an object for the proposition $(\mathsf{bright}(\mathsf{col}(\mathsf{p}_{38}))$. Using the theorem prover, the Cooperative Assistant succeeds in constructing such an object from the entries in his `Private` subcontext:

$$\mathsf{f}(\mathsf{p}_{38}) : \mathsf{bright}(\mathsf{col}(\mathsf{p}_{38}))$$

Because the question was a YES/NO-QUESTION, only the *existence* of a proof object matters. The Cooperative Assistant has found a proof object, hence the question will be answered affirmatively.

Had the user asked:

"Why is the colour of the pyramid bright?"

then the Cooperative Assistant would have been under the obligation to communicate how he came to believe $\mathsf{bright}(\mathsf{p}_{38})$. This is recorded in the proof object $\mathsf{f}(\mathsf{p}_{38})$; by the Gricean maxim of quantity the Cooperative Assistant should communicate only those ingredients of the proof that he does not believe to be already among the user's beliefs. By checking the `Common` context, the Cooperative Assistant can find out that the user already believes that there exists a pyramid $(\mathsf{p}_{38} : \mathsf{pyramid})$, and that it is small $(\mathsf{e}_5 : \mathsf{small}(\mathsf{p}_{38}))$. Hence, the Cooperative Assistant generates an answer from the only ingredient in his proof that is not in the `Common` context, viz. $\mathsf{f} : \Pi \mathsf{x}{:}\mathsf{pyramid}.\mathsf{bright}(\mathsf{col}(\mathsf{x}))$:

"Because every pyramid has a bright colour."

This answer is satisfactory since it provides the user with only the *new* information needed to infer how the Cooperative Assistant came to believe that the pyramid was bright.

In answering WH-QUESTIONS, other complications also occur. It may be particularly difficult to communicate the identity of an object, even if it occurs in the `Common` subcontext, because the message to the user should use the properties of the object to describe the object unambiguously for the user. Which description is most appropriate is a complicated matter, depending on aspects such as the difference in salience of the properties of the objects, previous utterances, and domain focus (see Cremers, 1995; 1997). We have not yet implemented the generation of such answers; in the current implementation the Cooperative Assistant will simply 'point out' the desired object by highlighting it on the screen.

5 Linguistic Interpretation and Dialogue Management

The user of the DenK system communicates in natural language with the Cooperative Assistant, who is responsible for the analysis of the user's dialogue contributions and the decision of what actions (domain actions and communicative actions) to perform. The design of the Assistant is based on the view that dialogue participants use language to perform communicative acts, aimed at changing the addressee's cognitive state in certain ways. Two basic aspects of a communicative act are its *semantic content*, describing the information the speaker[9] introduces into the dialogue, and its *communicative function*, describing the particular way in which the addressee's cognitive state is to be updated with the semantic content.[10] The communicative function determines, together with the semantic content, the effects of the communicative act on the Cooperative Assistant's information state.

The Assistant thus has to interpret the user's utterances in terms of their communicative function(s) and semantic contents, has to consider updating its information state accordingly, and has to generate communicative acts and/or domain acts.

5.1 Syntactic and Semantic Analysis

In linguistic analysis, a distinction is traditionally made between morphosyntactic, semantic and pragmatic analysis. Syntactic analysis is aimed at decomposing a complex expression into its constituent parts; this is the typical job of a parser (see e.g. Bunt & Tomita, 1996). Semantic analysis is concerned with the semantic consequences of the syntactic analysis in combination with the meanings of lexical items, and pragmatic analysis ties semantic analysis to aspects of the context of use. In the DenK project we have adopted the framework of *Head-driven Phrase Structure Grammar, HPSG* (Pollard & Sag, 1994) for linguistic analysis. This framework emphasizes the integration rather than the separation of these aspects of analysis; an HPSG parser does not build purely syntactic structures, but typed feature matrices which incorporate both syntactic, semantic, and pragmatic information.

Although presenting an integrated approach to the representation of syntactic, semantic, and pragmatic information, HPSG has been worked out primarily for syntactic analysis. Existing grammars in HPSG format have been developed to give detailed accounts of syntactic phenomena; for semantic phenomena this is much less so, and HPSG grammars that provide detailed treatments of pragmatic

---

[9] We use 'speaker' in a general sense here, to designate the participant whose contribution to the dialogue we are considering.

[10] The notions of 'semantic content' and 'communicative function' are obviously related to what in speech act theory are called 'propositional content' and 'illocutionary force'. For a definition of the concepts of 'communicative function' and 'semantic content' and a discussion of the difference with standard speech act concepts see Bunt (1997b).

phenomena, i.e. give an account of how syntactic and semantic phenomena depend on context information, have hardly been developed at all. We have added semantic and pragmatic components to the standard form of HPSG analysis that are well suited to the DenK system, in particular to the maintenance of formal representations of dynamic information states.

The interpretation of a natural language expression depends, in general, not only on the syntactic structure of the expression and the meanings of the constituent lexical items, but also on the context of use. While acknowledging this, existing theories of natural language interpretation are unable to take context information into account in a substantial way, in the absence of operational, articulate representations of context. In the DenK system we have such a representation, in the form of (a) a structured type-theoretical context recording the Cooperative Assistant's beliefs and the beliefs the Assistant takes to be shared with the user given the information exchange in the preceding dialogue; (b) the visual representation of objects and events in the application domain, as observable by the user and internally accessible by the Cooperative Assistant. The DenK system therefore provides an excellent basis for context-driven interpretation (and generation) of natural language utterances. We have designed such an interpretation process as consisting of two main stages:

1. A stage of context-independent interpretation, where the semantic consequences of morphosyntactic structure are made explicit;
2. A stage of context-dependent interpretation, where contextual information is taken into account in order to obtain contextually appropriate interpretations.

The first stage of this interpretation process produces representations in a format that is convenient for the second stage, which results in a type-theoretical representation (a CTT context). The intermediate format is called *'Underspecified Logical Form'*. ULF representations are allowed to be 'underspecified' in that they may leave open various aspects of the meaning of the natural language expression under consideration, such as the relative scopes of scope-bearing elements, the logical interpretation of natural language quantifiers, or the intended referents of anaphoric pronouns. (In fact, ULF representations leave open any aspect of meaning that cannot be decided on the basis of the syntactic evidence available in the utterance.) The ULF language has been designed by Kievit (1996), based on the representation language developed in the projects PLUS (Geurts & Rentier, 1993) and ΔELTA (see Rentier, 1993; Bunt, 1995b), and inspired in its general ideas by the Quasi-Logical Form language of the Core Language Engine (Alshawi, 1992).

An ULF representation of a natural language utterance is in fact a linearization of (parts of) an HPSG feature structure. The following example from the DK-2 application illustrates this. For the sentence *"Every button controls a lens"* the HPSG analysis by the DENK parser creates the following feature matrix:

$$
{}_{\text{ulf}}\begin{bmatrix}
\text{QUANT} & \text{event} \\
\text{VAR} & \boxed{1}\begin{bmatrix}\text{TNS} & \text{pres}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\text{restr}}\begin{bmatrix}
\text{RELN} & \text{controls} \\
\text{INST} & \boxed{1} \\
\text{ARGS} & \left\langle
{}_{\text{ulf}}\begin{bmatrix}
\text{QUANT} & \text{every} \\
\text{VAR} & \boxed{2}\begin{bmatrix}\text{GENDER} & \text{neuter} \\ \text{NUMBER} & \text{sing} \\ \text{PERSON} & \text{third}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\text{restr}}\begin{bmatrix}\text{RELN} & \text{button} \\ \text{INST} & \boxed{2} \\ \text{ARGS} & \langle\ \rangle\end{bmatrix} \right\}
\end{bmatrix},
{}_{\text{ulf}}\begin{bmatrix}
\text{QUANT} & \text{a} \\
\text{VAR} & \boxed{3}\begin{bmatrix}\text{GENDER} & \text{neuter} \\ \text{NUMBER} & \text{sing} \\ \text{PERSON} & \text{third}\end{bmatrix} \\
\text{RESTR} & \left\{ {}_{\text{restr}}\begin{bmatrix}\text{RELN} & \text{lens} \\ \text{INST} & \boxed{3} \\ \text{ARGS} & \langle\ \rangle\end{bmatrix} \right\}
\end{bmatrix}
\right\rangle
\end{bmatrix} \right\}
\end{bmatrix}
$$

Feature structures of type ulf introduce three features: *quant, var*, and *descr*. The value of *quant* is a quantifier which we take to represent quantification over an event or an object. Such events and objects correspond to a variable, the value of *var*, which intuitively also corresponds to a discourse referent in the sense of DRT. *Restr* takes as its values feature structures which have the features *reln* and *inst*, indicating a relation and an instance of this relation (the value of *inst* is token-identical to that of *var*). If the relation involves thematical dependents, then these are represented on a list called *args*, for 'arguments'. Since *args* takes values of type ulf, these typed feature structures are recursively defined.

The ULF for this utterance linearizes this feature structure as follows:

```
ulf(event,X,[tense:pres,inv:minus,voice:active,..],[control(X,[
     ulf(every,Y,[pers:3d,num:sg,gend:neuter],[button(Y,[])]),
     ulf(a,Z,[pers:3d,num:sg,gend:neuter],[lens(Z,[])]) ])])
```

It may be noted that all the words of the original sentence occur in this representation: not just the content words (*button, control, lens*), but also the quantifiers *every, a*. None of these words have been interpreted in this stage, and the relative scopes of the two quantifiers is not specified. The second stage of the interpretation process uses a (heuristically strengthened) variant of the Hobbs-Shieber (1987) scoping algorithm to take into account the syntactic restrictions on possible relative scopes.

5.2 Pragmatics and Dialogue Management

The second, context-dependent stage of interpreting user utterances, has the aim to compute 'contextually appropriate' interpretations, relating content words to the concepts of the application domain, relating pronominal anaphors and definite descriptions to the objects the user intended to refer to, interpreting natural language quantifiers in terms of logical relations appropriate for the application task and domain, etc. To be 'appropriate', a contextual interpretation should in general be compatible with all the context information available to the interpreter. Note that this does not imply that a contextually appropriate interpretation is necessarily unambiguous. The Cooperative Assistant is assumed to be able to perform appropriate actions on the basis of its understanding of the user's utterance (or rather, on the basis of the change of its information state that resulted from this understanding), so the essential requirement with respect to ambiguity is that the Assistant should construct interpretations that are specific enough to enable it to act. If the interpretation process is unable to construct such an interpretation, it will engage in a subdialogue with the user in order to interactively disambiguate the user's utterance.

The end result of interpretation is the construction of a representation in type-theoretical form, which in itself is unambiguous. Still, such a representation may cover more than one reading of a sentence. For instance, suppose the user asks: *"Every button controls a lens?"*. Besides the intuitively more plausible reading where *"every"* has scope over *"a"*, there is also a reading which can be paraphrased as: *"Is there a lens such that every button controls it?"*. The latter reading is a special case of the former: also in this case does every button control one or more lenses; it just so happens that the buttons all share the control of one of these lenses. This reading can thus be viewed as 'narrowing' the more general former reading; in logical terms, the latter ('stronger') reading implies the ('weaker') former. Both readings can be represented in CTT; in the DenK system, when a ULF representation allows several readings, one of which is more general than the others, we transform the ULF into the CTT representation of that reading - unless contextual evidence calls for a narrower reading (see Ahn et al., 1995).

Type-theoretical representations, as constructed by the context-dependent stage of the interpretation process, bear some resemblance to the representations of Discourse Representation Theory (Kamp & Reyle, 1993). The variable x in an introduction x: t can be seen as corresponding to a discourse referent in DRT, and the type as corresponding to a predicate in a DRT condition. The following example illustrates the possible use of CTT for semantic representation.

Consider the sentence *"A farmer laughs"*. The NP gives rise to an introduction of the form x: farmer, but farmer has to be introduced first to obtain a well-formed context. So we get, initially:

[farmer : class, x : farmer]

The VP corresponds to a predicate, applicable to farmers, and must be introduced as such, i.e. as a function from farmers to propositions:[11]

laugh : farmer $\rightarrow$ prop

We can now complete the CTT representation of the sentence by adding the statement that corresponds to the condition laugh(x) in DRT, to obtain the context:

[farmer : class, x : farmer, laugh : farmer $\rightarrow$ prop, e : laugh(x)]

This can be read as: farmers are objects (as opposed to propositions); x is a farmer; laughing is a property that farmers may have; there is evidence that x laughs. A CTT analysis of a sentence or a discourse thus introduces objects of various kinds and adds them to the context that grows incrementally as the discourse proceeds, similar to what happens in DRT. In fact, Ahn & Kolb (1991) have shown that CTT-contexts can be regarded as generalizations of DRSs.

During the contextual interpretation of utterances, one of the important things the Cooperative Assistant has to do is to link objects mentioned by the user to entities in the application domain. In the case of a definite reference, like *"the red block"*, the Cooperative Assistant looks for a specific red block that has been introduced in the dialogue before and therefore belongs to the Assistant's shared beliefs, or that can be detected in an unambiguous way in the current state of the discourse domain by inspecting its visual representation. In the case of an indirect reference, as in *"Move a red block"*, no specific object needs to be identified, and the choice of the object is left to the Assistant.

[11] For a systematic translation between DRT and CTT it is more convenient to construct somewhat more complex CTT representations, treating both nominal predicates (like *farmer*) and verbal predicates (like *laugh*) as function types (from *class* to *prop*). This would give rise to a CTT context like: [f_1 : class, farmer: Πx:class.prop, laugh: Πx:class.prop, pr_1: farmer(f_1), pr_2: laugh(f_1)].

New variables that arise from the introduction of definite and indefinite objects in the discourse are linked to entities in the domain by means of so-called 'satisfying assignments (Ahn & Kolb, 1991). For instance, the variable that results from interpreting *"the red block"* has to be linked to a suitable object in the Cooperative Assistant's shared beliefs; if the link can be established, the user can refer anaphorically to the object in subsequent utterances. If no such link can be established for instance because there is no such object, the Cooperative Assistant generates feedback to signal this.

Another crucial aspect of context-dependent interpretation concerns the assignment of communicative functions to user utterances. Formally, the interpretation of a user utterance results not just in a CTT expression, but in a pair of two expressions, each belonging to a well-defined formal language. Such pairs are called *annotated segments* (Piwek, 1995). The CTT expression in an annotated segment is a so-called 'CTT segment' and is defined as follows: if a nonempty CTT context Γ is split into two parts, Γ_1 and Γ_2, where Γ_1 is a legal context, then Γ_2 is a CTT segment. In other words, a segment represents certain information that is meaningful with regard to a given context. Annotations are sets of feature-value pairs that tell the system what to do with the information represented in the CTT segment. An annotation contains, for instance, the information whether the user's utterance is an assertion, a command, or a question. (This depends on syntactic properties of the utterance, as represented in its ULF, but also on the knowledge of the user's information state; see Beun, 1989.) Other information represented in an annotation is that certain variables, originating from anaphoric expressions, need to be bound. Using annotated segments, Beun and Kievit (1996) have developed a powerful algorithm for the contextual resolution of definite descriptions, taking into account the referential behaviour of participants in DenK-like situations as observed by Cremers (1995), which uses both the contextual information available in the CTT representation of the current information state and that available in the visual representation of the domain.

Having decided what to do with the semantic content of the user utterance, and extended its current information state accordingly, the Cooperative Assistant has to generate an adequate reaction. For that purpose, we consider the user's goals as established in the form of the communicative functions and the semantic content of the users's utterance. In the implemented DK-1 prototype we have assumed a simple and straightforward relation between an utterance's communicative functions and the underlying user goals: commands are used to change the state of the application domain and questions are used to obtain certain information about the domain. The eventual DK-system will incorporate more sophisticated pragmatic rules that take multifunctionality, indirectness, and dialogue control phenomena into account (cf. Beun, 1991; Bunt, 1994; 1995a).

The Cooperative Assistant considers two types of action in response to the user's communicative behaviour:

- domain acts, that are directed towards a change in the state of the domain;

– communicative acts, that are directed towards a change in the information state of the user (represented as the Cooperative Assistant's shared beliefs).

Domain acts are generated in reaction to a user command, such as *"Remove the blue block"*. In those cases, the Cooperative Assistant has to find the procedure that corresponds to the given command and that can be executed by the application.

The interpretation process can only be executed successfully if certain conditions are fulfilled. For instance, all content words in the user's utterances should be interpretable in terms of elements in that part of the Cooperative Assistant's information state that represents shared beliefs. If the user makes assumptions that are not part of the Assistant's beliefs (private or shared), these will be corrected by the Assistant, by means of corrective dialogue acts. For instance, to answer the question *"Why is the red block rotating?"* the Cooperative Assistant looks for a uniquely identifiable red block in the shared beliefs or in the visual domain model before the answer can be provided. If no such object is available, the Assistant generates a correction action. If the interpretation process is successful, this means in practice that the Cooperative Assistant can try to provide an answer to a user question, or to execute certain domain actions.

6 Visual Domain Modeling

In order to support multimodal interaction, an animation system that presents a convincing real-time view of the application domain must meet the following requirements:

1. interrogation of the domain status has to be allowed at any time, i.e. asynchronously with respect to the time evolution of the animation;
2. the display image has to be refreshed continuously, so that the user receives visual feedback irrespective of the state of the dialogue;
3. the user can at any time refer to elements of the visual model both via the Cooperative Assistant and via direct manipulation;
4. for generic support for a large variety of application domains:
   (a) instructions should be available to create objects[12] (both their geometrical shape and their autonomous motion behaviour) and to pass messages to objects to alter their properties and behaviour; objects also have to be able to pass messages to each other;
   (b) to facilitate programming complex behaviour, e.g., the motions of mechanical devices, of grasping an object, a library of versatile built-in motion methods must be available.

[12] We use terminology from object-oriented programming here. An 'object' is a variable containing data (its 'attributes') and optionally some program fragments (its 'methods'). Calling a method of an object is referred to as passing a 'message' to that object. An object is an instantiation of a 'class'; the class definition lists all available attributes and methods for objects in this class.

In order to meet these requirements, we have developed an architecture called the *Generalized Display Processor, GDP* (van Overveld, 1991); see Section 7.2. For programming the GDP, i.e. for defining the classes of objects inhabiting a domain, for creating such objects, for programming motion methods, for passing messages, etc. the language LOOKS (Language for Object Oriented Kinematic Simulations) has been defined and implemented (Peeters, 1994). LOOKS supports a variety of object-oriented features, such as data hiding, abstract data types, strong typing, genericity and multiple repeated inheritance; it also implements (quasi-)parallellism to facilitate the specification of concurrent motions. See further below, Section 7.2.

7 DENK System Architecture

According to the DENK 'triangle view', the DK-1 prototype system consists of two relatively independent components: the Cooperative Assistant and the Application Model. We outline the design each of these components in the next two sections. The architecture of the eventual DK-2 system will be an elaboration of this design.

7.1 The Cooperative Assistant

The heart of the DK-1 system is formed by the software components implementing the Cooperative Assistant, shown in Fig. 2. These are the following:

`Parser`: A parser for natural language, with an implemented natural language fragment.[13] After considering a number of alternatives, the publicly available *Attribute Logic Engine (ALE)* (Carpenter, 1994) was chosen for HPSG-based analysis. The standard HPSG analysis was modified so that the parser delivers underspecified logical form representations. This is only a modification in representation *format*; the 'Semantics Principle' of HPSG, that accounts for the construction of semantic representations during constraint-based parsing, has only been modified so as to work with the modified representations, but has otherwise remained unchanged.

The implemented ULF language forms the interface between the parser and the Pragmatic Rule Interpreter.

`Pragmatic Rule Interpreter`: A variety of 'pragmatic' rules of various kinds for aspects of context-dependent interpretation (quantifier scoping, referent determination, etc.), for updating the system's information state, and for generating appropriate (re-)actions, needed by the Cooperative Assistant

[13] In the DK-1 system, the implemented natural language fragment is a toy fragment of Dutch. For the DK-system under construction, the form of natural language is typed English with certain syntactic, semantic and pragmatic limitations reflecting the properties of the tasks and the domain of the application (see Verlinden, 1996).

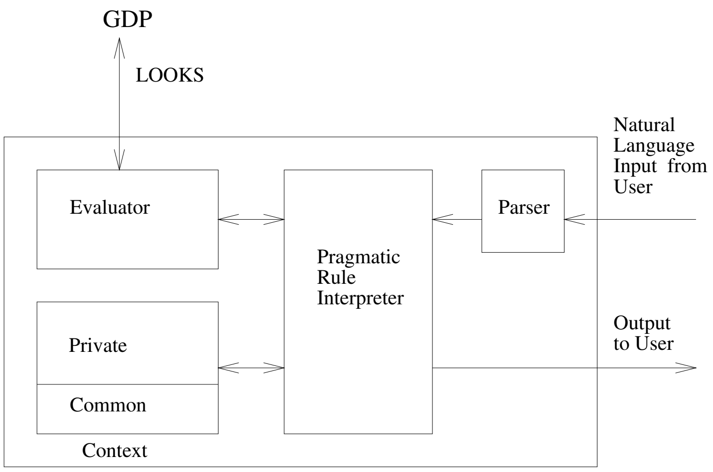

Fig. 2. Architecture of the Cooperative Assistant in the DK-1 system.

for its operation. In the DK-1 prototype system, only very small sets of rules have been implemented, and the application of these rules is performed in a rather ad hoc manner by the `'Pragmatic Rules Interpreter'` component (see also below, the section Conclusions and Future Work).

For using and updating context information, represented in CTT, and handling information extracted from user utterances and expressed in CTT, a proof checker and a reasoning system for type theory have been implemented. The proof checker checks whether a type-theoretical context is well-formed, in particular whether all expressions in the context under consideration are correctly typed. This checker is based on theoretical work in type theory; see e.g. Barendregt (1991). The reasoning system for type theory, based on the work of Helmink and Ahn (1991), incorporates strategies for combining type-theoretical deduction steps in order to facilitate effective reasoning with modal information. (In the DK-1 system, only the epistemic modalities of (simple) belief and believed shared belief are considered; in the DK-2 system, temporal modalities will also be handled.)

An implemented 'Conceptual Lexicon' relates ULF predicates to the predicates and types defined in the CTT model of the DK-1 domain. ULF predicates have a straightforward correspondence to natural language content words; the Conceptual Lexicon indirectly relates these words to the concepts of the application domain.

`Context:` A representation of the Cooperative Assistant's information state in the form of a CTT context. Two epistemic modalities are distinguished in this representation, involving two agents: belief and assumed shared beliefs. The CTT context as a whole represents the beliefs of the Cooperative Assistant; those beliefs that the Assistant believes to be shared with the user form the subcontext `Common`; the other beliefs form the subcontext `Private`. This representation is based on theoretical work in type theory by Borghuis (1994).

A model of the DK-1 blocks world has been formalized in Constructive Type Theory and implemented. This model specifies the classes of objects in the domain (pyramids, cubes, colours,..), the predicates that apply to these objects (colour, size, motion, brightness,..), and the constraints that hold in the domain, e.g. the fact that all pyramids have bright colours (see the example context in Section 4.2). This model cannot be changed in a dialogue, and since the user is assumed to know the application domain, it forms a static part of `Context`. It also seems plausible to assume that the user knows the concepts (object classes, predicates) of the application, hence this information forms part of `Common`.

`Evaluator:` An evaluator for converting an expression in type theory into LOOKS (see the next section) and for accessing the visual domain model in order to compute the values of these expressions, given the current state of the domain as represented by the GDP. This involves translating type-theoretical expressions into combinations of the application domain primitives used by the GDP.

The Cooperative Assistant's communicative behaviour is controlled by the `Pragmatic Rule Interpreter`, which produces a simple form of cooperative behaviour using three types of contextual information:

1. the information state represented in `Context`;
2. the most recent (communicative) action performed by the user (this act is kept on a local stack, accessible to the `Pragmatic Rule Interpreter`);
3. the current state of the application domain as represented by the GDP.

First, the interpreter analyses the user's utterance within the current dialogue context and, if no communication failures are noted, it updates `Context` with the new information in a way determined by the communicative function of the utterance. If the communicative function of the utterance is a command, a domain-related action is performed; if the function is a question, the question is answered, or appropriately responded to if no straight answer can be given. Since the Cooperative Assistant is considered to be an expert about the domain, he does not accept new domain information from the user, and interprets declarative statements about the domain as (verification) questions (see Beun, 1989).

7.2 The Generalized Display Processor (GDP)

The GDP, implementing the Application Model, is a generic emulator for visual domain representation, which is capable of simulating a domain given a formal definition in LOOKS. For its architecture, the GDP can be viewed as a virtual processor consisting of a parser, an interpreter, rendering support, and a database.

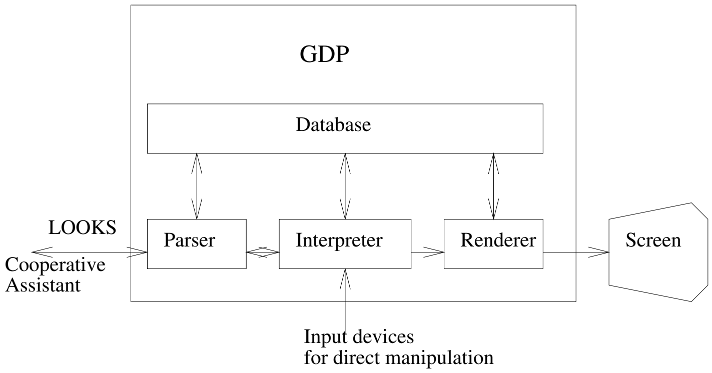

Fig. 3. The Generalized Display Processor.

The parser provides for the communication with the Cooperative Assistant; it checks the incoming information and, if necessary, sends messages back from the interpreter to the Cooperative Assistant. The interpreter calculates the values of the attributes of the objects (e.g. position, speed) in order to generate a new frame of animation, which is displayed on screen by means of the renderer.[14] The behaviour of the objects may be autonomous, or dictated by the Cooperative Assistant or by input devices for direct manipulation (e.g. the mouse).

At any discrete time, a complete description of all geometrical attributes of a moving scene is stored in the database. The database consists of two tables: the class table and the object table. The class table contains the class definitions that state which attributes exist for a particular class as well as which methods it may execute.[15] The object table contains the actual objects in the application domain, i.e. instantiations of existing classes.

At any time, fragments of LOOKS may be passed to the parser. Such a fragment can be either a class definition, an object definition or a message. If this

[14] Provided the time granularity of the animation is sufficiently small and the renderer is sufficiently fast, the impression of a moving display results.

[15] A class may be defined from 'scratch' or it may inherit attributes and/or methods from other classes. The types of its attributes, as well as the types of the formal parameters in its method headings and the return types of its methods may either be earlier defined classes or generic types.

fragment is successfully parsed, and if it was a

class definition, then an entry is created in the class table. Apart from the user-defined classes, LOOKS supports a variety of pre-defined classes, including integers, reals, vectors, movable geometric objects, cameras, (coloured) light sources, windows, etc.;

object definition, then an entry is created in the object table;

message to an object, then the corresponding method is handed to the interpreter to be executed.

To enable interaction between the Cooperative Assistant and the GDP (see Fig. 3), a close relation must be established between LOOKS and the type-theoretical formalism used by the Assistant. Type-theoretical expressions are virtually semantically grounded in LOOKS, and the translation from CTT to LOOKS comes down to a standard evaluation process,[16] where complex type-theoretical expressions are expanded into basic expressions that can be interpreted in LOOKS.

In order to generate a frame of animation, the interpreter installs a method it receives from the parser into the list of active methods and executes the methods it encounters. Most active methods can be fully executed (e.g. assignments, expression evaluations, object transformations), after which they are removed from the list.

Methods may, however, also contain a 'synchronize' statement (typically within the body of an (infinite) loop). Execution of an active method proceeds until it is either fully executed, or until a 'synchronize' is encountered. In the latter case, execution stops, but the method is kept in the list; it will again receive the interpreter's attention when preparing the next frame. Since several methods may contain a 'synchronize', several of which may be kept in the active method list simultaneously, the synchronize mechanism may serve to achieve quasi-parallelism.

Also, the execution of a method, due to passing an asynchronous message, may invoke the execution of another method by calling this method. The latter method is also put into the list of active methods, and it will be executed as well during the preparation of the frame.

If no more active methods can proceed any further, the preparation of the frame is complete, and a snapshot of the objects is rendered into the active window(s) with respect to the active virtual camera(s), making use of the active light sources and simulated material properties (colour, shininess, etc.).

Part of the preparation of each frame is also the taking into account of user interaction. Via mouse events and appropriate LOOKS system methods, the user may interact with the ongoing animation (e.g. selecting an object and calling the mouse notifier method that has been assigned to this object).

The current GDP implementation is designed to run on a high-end graphics workstation where it produces a flicker-free display of shaded images, illuminated by simulated light sources, consisting of several hundreds of polygons with

[16] See also the `'Evaluator'` in Fig. 2.

a frame update rate between 15 and 20 frames/sec. It covers most of the requirements and functionality as described above; direct manipulation interaction tools and versatile built-in motion methods are presently under construction.

8 Conclusions and Future Work

Following an 'inside-out' strategy, the DENK project has first developed a formal model of communication in terms of changing cognitive states and an implementation of this model in a first prototype system. The formalization in constructive type theory of the context-change model of communication, of which the conceptual basis is provided by Dynamic Interpretation Theory, opens up a way to make progress in dealing with context as one of the major factors contributing to the determination of the meaning of communicative behaviour.

The DENK-approach to user interface design enables one to formulate and to implement rules for cooperative behaviour, independent of a particular application domain or a particular mix of communicative modalities. The DENK design originates from the desire to establish *natural communication* between the application domain and its user, largely independent of the surface forms of messages. Well-known phenomena in natural language semantics and pragmatics, such as the context dependence of messages, the use of definite and indefinite reference, the Gricean maxims for cooperative linguistic behaviour, and the use of deixis, can be shown to follow naturally from the fundamental properties of communication viewed as intended change of context. Even the relatively simple model of communication and context implemented in the DK-1 prototype system, presented in this chapter, can be useful in dealing with such phenomena, as the work on resolving definite expressions by Beun & Kievit (1996) demonstrates.

The DK-1 system is primarily interesting as the implementation of a formal backbone of a generic cooperative communication system, with its powerful approach to context modeling in type theory connected to representation formalisms for natural language semantics and visual domain representation. Future (and current) work building on this basis goes in three directions:

Continuation of the inside-out strategy. Moving more to the outer sides of interaction, a substantial and useful fragment of English, based on empirical studies (such as Cremers, 1995) will be defined with corresponding ULF extensions. A set of direct manipulation acts will be specified to allow the user to act directly on the (visual representation of the) application domain, with consequences that the Cooperative Assistant notes by internal inspection of the domain.

Perhaps the most crucial step in outward direction to be taken, to achieve a basis of truly cooperative multimodal interaction, is the development of the pragmatic rules and procedures for their application in the Cooperative Assistant. This is in particular the case with the rules used the Assistant to generate communicative acts, and with the rules and procedures for interpreting combined use of input modalities by the user. The work by Beun and

Kievit on reference resolution (Beun & Kievit, 1996) forms a first step in this direction. Continued work in this area is expected to result in a structuring of the `Pragmatic Rule Interpreter` into several submodules, interacting using annotated segments as interface language.

Inward refinement. The CTT, ULF, and LOOKS formalisms provide a solid basis for developing cooperative human-computer interfaces, but are still in need of improvement. The CTT formalism as developed in the DenK project extends standard type theory with two epistemic modalities associated with two agents (simple belief and believed shared belief), which is the minimal epistemic variation needed to account for context-driven information exchange. To account for more subtle forms of information exchange, additional epistemic modalities have to be considered. Perhaps more urgently, the time dependence of elements in belief contexts will be taken into account in an enhanced version of CTT.

Continued implementation of the inside-out strategy. The extensions of CTT just mentioned (as well as some others) will be implemented in the DK-2 system, affecting `Context` as well as the reasoning and proof checking procedures used by the Cooperative Assistant.

The domain of electron microscope operation will be formalized in CTT and implemented both in the `Context` and `Evaluator` components in the Cooperative Assistant, as in LOOKS and the GDP.

Implementation of the fragment of English defined for this domain (both for user input and for Cooperative Assistant output), in combination with the direct manipulation possibilities offered to the user, will give the system and the user the interactive modalities required for making full use of the implementation of the 'triangle view' of natural interaction with a simulated application domain.

Acknowledgements

We would like to thank the organization for inter-university cooperation between the universities of Tilburg and Eindhoven ("Samenwerkingsorgaan Brabantse Universiteiten") for providing funds to make the DenK project possible. The research reported here is carried out in the Language Technology and Artificial Intelligence Group of the Faculty of Arts at Tilburg University, in the Department of Mathematics and Computer Science at Eindhoven University of Technology, and at the Center for User-System Research IPO in Eindhoven. The participating institutions have all supported the DenK project financially and in other ways.

The following people have contributed to the DenK project: René Ahn, Robbert-Jan Beun, Gino van den Bergen, Tijn Borghuis, Anita Cremers, Frens

Dols, Jan Jaspars, Leen Kievit, Paul Piwek, Eric Peeters, Gerrit Rentier, and Margriet Verlinden (mostly in PhD projects), and Rita Roelfs and Liesbeth Melby in providing programming support. Supervision of parts of the project has been given by Jos Baeten, Don Bouwhuis, Rob Nederpelt, and Kees van Overveld; Harry Bunt is overall project leader.

A word of special thanks goes to Hugo van Leeuwen from PHILIPS Electron Optics, for most valuable support in developing the electron microscope training simulation application of the DenK system.

References

Ahn, R. (1995a) Communicating Contexts, a Pragmatic Approach to Information Exchange. In: P. Dybjer, B. Nordström, & J. Smith (eds.) *Types for Proofs and Programs: Selected Papers.* Berlin: Springer, 1–13.

Ahn, R. (1995b) Logical Model of the Electron Microscope. *DenK Report* 95/15, Tilburg University.

Ahn, R. (1996) Monotonic description of dynamic worlds. In: *Proc. NIAC'96 Conference* (forthc.)

Ahn, R. & Kolb, H.P. (1990) Discourse Representation meets Constructive Mathematics. In: L. Kálmán & L. Pólos (eds.) *Papers from the Second Symposium on Logic and Language*, Akademiai Kiadó, Budapest, 105–124.

Ahn, R., Kievit, L., Rentier, M. Verlinden, M. (1995) Two Levels of Semantic Representation in DenK. In: T. Andernach, M. Moll & A. Nijholt (eds.) *Papers from the Fifth CLIN Meeting.* Universiteit Twente, 35–58.

Ahn, R. & Borghuis, T. (1996) Chronicle. *DenK Report* 96/21, Tilburg University.

Allwood, J. (1976) *Linguistic Communication as Action and Cooperation.* Gothenburg Monographs in Linguistics 2, University of Göteborg.

Allwood, J. (1997) An Activity-Based Approach to Pragmatics. In: W.J. Black & H.C. Bunt (eds.) *Abduction, Belief and Context, Studies in Computational Pragmatics*, University College London Press (forthc.)

Alshawi, H. (1992) *The Core Language Engine*, Cambridge: Cambridge University press.

Austin, J.L. (1962) *How to do things with words.* Oxford: Oxford University Press.

Barendregt, H. (1991) Introduction to Generalized Type Systems. *Journal of Functional Programming, 1(2).* 125-154.

Barendregt, H. (1992) Lambda calculi with types. In: S. Abramsky, D. Gabbay and T. Maibaum (eds.) *Handbook of Logic in Computer Science.* oxford: Oxford University Press.

Beun, R.J. (1989) *The Recognition of Declarative Questions in Information Dialogues.* PhD Dissertation, Tilburg University.

Beun, R.J. (1991) A Framework for Cooperative Dialogues. In: M.M. Taylor, F. Néel & D.G. Bouwhuis (eds.) *Proceedings of the Second Venaco Workshop on the Structure of Multimodal Dialogue.* Maratea, Italy.

Beun, R.J. & Kievit, L.A. (1996) Resolving definite expressions in dialogue. In: *Proc. Fifth Int. Conference on Pragmatics*, Mexico City, July 1996.

Borghuis, T. (1993) Interpreting Modal Natural Deduction in type theory. In: M. de Rijke (ed.) *Diamonds and Defaults.* Kluwer Academic Publishers. 67-102.

Borghuis, T. (1994) *Coming to Terms with Modal Logic: On the Interpretation of Modalities in Typed λ-Calculus.* Ph.D. Dissertation, Eindhoven University of Technology.

De Bruijn, N.G. (1980) A survey of the project Automath. In: Seldin & Hindley (eds.) *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms.* Academic Press. 579-606.

Bunt, H.C. (1989) Information dialogues as communicative action in relation to partner modeling and information processing. In: M.M. Taylor, F. Néel & D.G. Bouwhuis (eds.) *The Structure of Multimodal Dialogue.* North-Holland, Amsterdam. 47-73.

Bunt, H.C. (1991) DIT - Dynamic Interpretation in Text and Dialogue. In: L. Kálm'an & L. Pólos (eds.) *Papers from the Second Symposium on Logic and Language*, Budapest: Akademiai Kiadó, 67–104.

Bunt, H.C. (1994) Context and Dialogue Control. *THINK Quarterly 3(1)*, 19-31.

Bunt, H.C. (1995a) Dialogue Control Functions and Interaction Design. In R.J. Beun, M. Baker, and M. Reiner (eds.) *Dialogue and Instruction: Modelling Interaction in Intelligent Tutoring Systems.* Berlin: Springer, 197–214.

Bunt, H.C. (1995b) Dialogue Semantics and Pragmatics in the ΔELTA Project. In: L. Dybkjaer (ed.) *Proc. of the Second on Spoken Dialogue and Discourse Workshop*, Roskilde University, 1–27.

Bunt, H.C. (1997a) Dialogue Context Modeling. In: *Proc. CONTEXT'97, International Interdisciplinary Conference on Modeling and Using Context*, Rio de Janeiro, February 1997, 130–150.

Bunt, H.C. (1997b) Iterative Context Specification and Dialogue Theory. In: W.J. Black & H.C. Bunt (eds.) *Abduction, Belief and Context, Studies in Computational Pragmatics*, London: University College London Press (forthc.)

Bunt, H.C. Ahn, R., Kievit, L.A., Piwek, P., Verlinden, M., Beun, R.J., Borghuis, T. & Overveld, C. van (1998) Cooperative Dialogue with the Multimodal DenK System. In: H.C. Bunt, R.J. Beun, T. Borghuis, L.A. Kievit & M. Verlinden (eds.) *Proc. Second International Conference on Cooperative Multimodal Communication, CMC/98*, Tilburg: Tilburg University, 44–62.

Burger, J.D. & Marshall, R.J. (1993) The application of natural language models to intelligent multimedia. In: M. Mayburry (ed.) *Intelligent Multimedia Interfaces.* Massachusetts: MIT Press, 174–196.

Carpenter, R. (1982) *The Logic of Typed Feature Structures.* Cambridge: Cambridge University Press.

Coquand, T. (1985) *Une théorie des constructions.* Thèse de troisième cycle. Paris: Université de Paris VII.

Cremers, A.H.M. (1995) *Reference to Objects: An Empirical Investigation of Task-oriented dialogues.* PhD Dissertation, Eindhoven University of Technology.

Cremers, A.H.M. (1997) Object Reference in Task-oriented Keyboard Dialogues. *This volume.*

Curry, H.B. & Feys R. (1958) *Combinatory Logic, Vol. 1*, North Holland Publishing Company.

Geurts, B. & Rentier, G. (1993) The PLUS Grammar. Internal Report ESPRIT Project PLUS, Institute for Language Technology and Artificial Intelligence ITK, Tilburg.

Helmink, L. & Ahn, R. (1991) Goal-oriented proof construction in type theory. In: G. Huet and G. Plotkin (eds.) *Logical Frameworks.* Cambridge: Cambridge University Press, 120–148.

Hobbs, J.R. & Shieber, S.M. (1987) An algorithm for generating quantifier scopings. *Computational Linguistics* 13 (1–2), 47–63.

Hutchins, E. (1989) Metaphors for Interface Design. In: M.M. Taylor, F. Néel & D.G. Bouwhuis (eds.) *The Structure of Multimodal Dialogue.* Amsterdam: North-Holland, 11–28.

Jaspars, J.O.M. (1993) *Calculi for Constructive Communication.* Ph.D. Dissertation, Institute for Language Technology and Artificial Intelligence ITK, Tilburg.

Kamp, J.A.W. & Reyle, U. (1993) From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. Dordrecht: Kluwer Academic Publishers.

Kievit, L.A. (1996) ULF. DenK Report 96/27, Tilburg University.

Martin-Löf P. (1984) *Intuitionistic Type Theory*. Naples: Bibliopolis.

Neal, J.G. & Shapiro, S.C. (1991) Intelligent multimedia interface technology. In: J.W. Sullivan & S.W. Taylor (eds.) *Intelligent User Interfaces.* Reading, MA: Addison-Wesley, 11–43.

Overveld, C.W.A.M. van (1991) The Generalized Display Processor as an approach to real time interactive 3-D computer animation. *The Journal of Visualisation and Computer Animation, 2(1)*, 16–21.

Peeters, E.A.J. (1994) *Design of an object-oriented interactive animation system.* PhD Dissertation, Eindhoven University of Technology.

Pollard, C. & Sag, I. (1994) *Head-driven Phrase Structure Grammar.* University of Chicago Press.

Rentier, G. (1993) ULF. ΔELTA *Working Paper, May 1994*, ITK, Tilburg University.

Searle, J.R. (1969) *Speech Acts.* Cambridge: Cambridge University Press.

de Souza, C.S. (1993) The semiotic engineering of user interface languages. *International Journal of Man-Machine Studies, 39*, 753–774.

Verlinden, M. (1996) Development of the cores syntax for DenK. DenK Report 96/22, Tilburg University.

Wahlster, W., André, E., Finkler, W., Profitlich, H.-J. & Rist, T. (1993) Plan-based integration of natural language and graphics generation. *Artificial Intelligence, 63*, 387–427.

Wilson, M.D., Sedlock, D., Binot, J-L. & Falzon, P. (1991) An architecture for multimodal dialogue. In: M.M. Taylor, F. Néel & D.G. Bouwhuis (eds.) *Proceedings of the Second Venaco Workshop on the Structure of Multimodal Dialogue*. Maratea, Italy.