Tagging Communication Problems in Spoken Dialogue Systems: On-line or Off-line?

Laila Dybkjær and Niels Ole Bernsen
Natural Interactive Systems Laboratory, University of Southern Denmark
Science Park 10, 5230 Odense M, Denmark
laila@nis.sdu.dk, nob@nis.sdu.dk

Abstract

The paper describes our work on tagging communication problems in human-computer shared-goal spoken dialogues. The coding scheme is not otherwise restricted to a particular task or domain. The tag-set is rooted in a set of co-operativity guidelines. Tagging is done off-line at design time. The purpose is to discover problems early in the dialogue development process in order to prevent them from occurring in later versions of the dialogue. We describe the coding scheme and its application and discuss the issue of on-line versus off-line tagging of communication problems.

1 Introduction

The increasing number of advanced spoken language dialogue systems (SLDSs) which support people in carrying out ordinary tasks, such as flight/train timetable consultation, ticket booking, or directory inquiry, demands rigorous methods and tools for identifying, analysing, preventing, and repairing problems in spoken human-machine interaction. Annotation of communication problems in spoken dialogue corpora not only helps developers and researchers extract information on the deficiencies of emerging dialogue interaction models, but may also yield clues as to how these might be improved.

Communication problems, if detected by users, typically lead to clarification or repair meta-communication. This is usually not really a problem in human-human dialogue. However, with current SLDS technology the possibility of real-time handling of clarification and repair meta-communication is still seriously limited. In particular, user needs for clarification meta-communication that arise from the way the system addresses its domain, can easily surpass the system's meta-communication skills.

We were faced with exactly this kind of problems while designing, implementing, and testing the dialogue model for the Danish dialogue system (Bernsen et al. 1998) which was a Danish domestic flight ticket reservation system. In the process of analysing collected (Wizard of Oz-simulated) human-computer dialogues, we developed a coding scheme for markup of communication problems. This scheme facilitated and made more systematic our analysis of dialogues and the resulting proposals for dialogue model improvements. In Section 2 we describe the background for, and the development of, the communication problems coding scheme. Section 3 presents the coding scheme itself. Section 4 discusses applications of the coding scheme. Section 5 concludes the paper by discussing the issue of on-line versus off-line tagging.

2 Background of the Coding Scheme

Communication problems are different in several respects from most other phenomena that are usually annotated and studied in a spoken dialogue corpus. Most notably, communication problems need not necessarily be present in a corpus at all. In fact, the fewer there are, the better. This is in direct contrast with, e.g., prosodic and morpho-syntactic phenomena, or dialogue acts, which are present in any spoken dialogue corpus. To a large extent, the same is true for co-reference. All these phenomena are among the building blocks of spoken dialogue. Communication problems, on the other hand, are disruptive to the dialogue and co-operative human interlocutors usually try to avoid them. In particular for SLDSs, co-operative system communication is important for avoiding communication problems which often lead to user dialogue behaviour with which the system cannot cope.

2.1 Existing Coding Schemes

Annotation of communication problems is clearly in its infancy. In fact, we are not aware of other coding schemes which are particularly tailored to the identification and description of communication problems. However, aspects of communication problems are quite often reflected in coding schemes for other levels of spoken dialogue. Such schemes do not have a direct focus on communication problems. Rather, they include phenomena that relate to the level in focus, e.g. co-reference or dialogue acts, as well as to communication problems, cf. (Mengel et al. 2000). This is perhaps not surprising given the cross-level nature of communication problems. Thus, some communication problems are caused by flawed grammar or vocabulary design, i.e., errors at the morpho-syntactic level. Other problems may be due to misinterpretation or non-interpretation of co-reference, and so on.

2.2 Development and Test of the Coding Scheme for Communication Problems

The coding scheme for communication problems to be discussed in this paper grew out of our analysis of Wizard of Oz-simulated human-computer dialogues. The aim of the analysis was exactly to detect communication problems and, on this basis, to improve the dialogue model. As a bi-product, the coding scheme was developed as described in the following.

The dialogue model of the Danish dialogue system mentioned in Section 1 was developed using the Wizard of Oz (WOZ) simulation method. In the process of analysing the collected dialogues we established a set of guidelines for the design of cooperative spoken dialogue. Each observed problem was considered a case in which the system, in addressing the user, had violated a guideline of cooperative dialogue. The WOZ corpus analysis led to identification of 14 guidelines of cooperative spoken human-machine dialogue based on analysis of 120 examples of user-system interaction problems. If those guidelines were observed in the design of the system's dialogue behaviour, we assumed, this would increase the smoothness of user-system interaction and reduce the amount of user-initiated meta-communication needed for clarification and repair.

During the development of our 14 co-operativity guidelines, we were not aware of the potential relevance of Grice's work. Upon discovering that relevance, the guidelines were refined and consolidated through comparison with Grice's well-established body of maxims of cooperative human-human dialogue which turned out to form a subset of our guidelines (Grice 1975, Bernsen et al. 1996). The resulting 22 guidelines were grouped under seven different *aspects* of dialogue, such as informativeness and partner asymmetry, and split into *generic* guidelines and *specific* guidelines, cf. Figure 1. The specific guidelines are a refinement of the generic ones, and are thus subsumed by the latter. The generic guidelines are more general and express what to do or take into account when communicating. The specific guidelines specialise the generic guideline by which they are subsumed to certain classes of phenomena, explain how to do something expressed by the generic guideline, and are specifically aimed at system design. Although subsumed by generic guidelines, the specific guidelines are important in interaction design because they serve to elaborate on the kind of interaction model that the developer should be looking for when designing co-operative system dialogue behaviour.

It should be noted that not every generic guideline subsumes some specific guideline(s), and the specific guidelines do not add up to, or replace, the generic guideline by which they are subsumed. A given communication problem should always be described, if possible, by referring to the violation of a specific guideline if there is one which fits. Otherwise, the reference should be to a generic guideline.

The consolidated guidelines were then tested by using them in the diagnostic evaluation of a corpus of 57 dialogues collected during a scenario-based, controlled user test of the implemented system.

Dialogue Aspect	GG No.	SG No.	Generic or Specific Guideline
Group 1 **Informativeness**	GG1		*Make your contribution as informative as is required (for the current purposes of the exchange).
		SG1	Be fully explicit in communicating to users the commitments they have made.
		SG2	Provide feedback on each piece of information provided by the user.
	GG2		*Do not make your contribution more informative than is required.
Group 2 **Truth and evidence**	GG3		*Do not say what you believe to be false.
	GG4		*Do not say that for which you lack adequate evidence.
Group 3 **Relevance**	GG5		*Be relevant, i.e. be appropriate to the immediate needs at each stage of the transaction.
Group 4 **Manner**	GG6		*Avoid obscurity of expression.
	GG7		*Avoid ambiguity.
		SG3	Provide same formulation of the same question (or address) to users everywhere in the system's dialogue turns.
	GG8		*Be brief (avoid unnecessary prolixity).
	GG9		*Be orderly.
Group 5 **Partner asymmetry**	GG10		Inform the dialogue partners of important non-normal characteristics which they should take into account in order to behave cooperatively in dialogue. Ensure the feasibility of what is required of them.
		SG4	Provide clear and comprehensible communication of what the system can and cannot do.
		SG5	Provide clear and sufficient instructions to users on how to interact with the system.
Group 6 **Background knowledge**	GG11		Take partners' relevant background knowledge into account.
		SG6	Take into account possible (and possibly erroneous) user inferences by analogy from related task domains.
		SG7	Separate whenever possible between the needs of novice and expert users (user-adaptive dialogue).
	GG12		Take into account legitimate partner expectations as to your own background knowledge.
		SG8	Provide sufficient task domain knowledge and inference.
Group 7 **Repair and clarification**	GG13		Enable repair or clarification meta-communication in case of communication failure.
		SG9	Initiate repair meta-communication if system understanding has failed.
		SG10	Initiate clarification meta-communication in case of inconsistent user input.
		SG11	Initiate clarification meta-communication in case of ambiguous user input.

Figure 1. Guidelines for cooperative system dialogue. GG means generic guideline. SG means specific guideline. Generic guidelines are expressed at the same level of generality as are the Gricean maxims (marked with an *). Each specific guideline is subsumed by a generic guideline. The left-hand column characterises the aspect of dialogue addressed by each guideline.

The fact that we had the scenarios at hand meant that problems of dialogue interaction could be objectively detected through comparison between expected (according to the scenario) and actual user-system exchanges. Each detected problem was (a) characterised with respect to its symptom, i.e. the spoken dialogue exchange that demonstrated that something was amiss; (b) a diagnosis of the problem was made, sometimes through inspection of the log of system module communication; and (c) one or several cures for the problem were proposed. The ‘cure’ part of diagnostic analysis suggests ways of repairing the system’s dialogue behaviour. In addition, the diagnostic analysis may demonstrate that a new guideline for cooperative dialogue design must be added to the guidelines set, thus enabling continuous assessment of the scope of the tag set.

We found that nearly all communication problems in the user test could be classified as violations of our guidelines. Two specific guidelines on meta-communication, SG10 and SG11 (Figure 1), had to be added, however. This was no surprise as meta-communication had not been simulated and thus was mostly absent in the WOZ corpus.

2.3 Scope of the Coding Scheme

The NISLab annotation scheme cannot be claimed to account for all possible communication problems. It was created with the purpose of improving spoken language dialogue system design and, so far, it has only been tested on dialogues which were:

- shared-goal,
- human-computer, and
- two-participant dialogue.

Thus, the NISLab scheme is not claimed to be valid for human-human dialogues and non-shared goal dialogues since it has not been tested for these conditions.

Shared-goal dialogues are dialogues in which the interlocutors collaborate to achieve a common goal, such as booking a ferry ticket or getting/ providing information about flight arrivals. Generally speaking, today’s spoken language dialogue systems are shared-goal systems which take for granted that the user’s goal is to carry out (one of) the task(s) that the system can support. Shared-goal dialogue may be contrasted with general conversation which is subject to possible conflicting goals and intentions among the participants.

Human-human dialogue has many more facets than today’s spoken *human-machine* dialogue is capable of handling. Therefore, we cannot discount the possibility that there are communication problems in human-human dialogue which might call for additional guidelines for co-operative dialogue behaviour compared to those in Figure 1. Human-human communication problems may, for instance, derive from conflicting goals/intentions, talking ‘above one’s head’, lying, and hidden agendas, not to speak of the full gamut of natural interactivity issues, such as facial or gesture communication misunderstandings. Whether such potential problems in understanding between humans in dialogue can be captured by the annotation scheme developed and described here is still an open question. However, extending the current set of guidelines will be easy to do by using the coding module presented in Section 3.

So far, the coding scheme has been tested only on *two-participant dialogues*. Spoken human-computer dialogue is normally between one human and one machine but multi-party human-human-machine dialogue is now advancing towards the top of the research agenda world-wide. It may seem likely that the communication problems will also apply to multi-party human-human-machine dialogues but this remains to be tested.

The primary focus of our studies so far has been to mark up communication problems caused by the system because the emphasis was on investigating how system interaction could be improved to achieve a smooth dialogue with users. Of course, users also commit errors from time to time which can be the direct cause of a communication problem. User errors have only been investigated to a limited extent in this connection, i.e., in their own right (Dybkjær et al. 1998a) and we still lack detailed knowledge of their mechanisms. We are mainly interested in those cases of user errors that are triggered by inappropriate system interaction. However, it seems likely that the human interlocutor is able to cause the same categories of communication problems as the system does, i.e. by violating the guidelines listed in Figure 1.

3 Coding Scheme Description

Communication problems span a wide range of phenomena. They refer either to (1) some item of information which was omitted, (2) to a single word causing problems, (3) to several words, (4) a whole utterance, (5) several utterances (or turns), or even, in principle, (6) more than one dialogue which led to the miscommunication. In practice, communication problems most frequently refer to the first four of the options mentioned. Furthermore, the markup of communication problems is not always non-contiguous but may be contiguous or even overlap in various ways.

Mark-up of communication problems normally references an orthographic transcription and a file containing types of violations. The latter is edited along with the coding. The types of violation again reference the guidelines for cooperative dialogue, cf. Figure 2.

Communication problems are tagged as types of violation of the guidelines for co-operative spoken dialogue. A particular guideline may be violated in several different ways. For example, GG7 (avoid ambiguity) would be violated by not saying whether the time "9 o'clock" given to the user by the system means 9 am or 9 pm. Another type of violation of the same guideline might occur if it was not made clear whether a certain flight arrival time refers to that given by the timetable or to the actual expected arrival time.

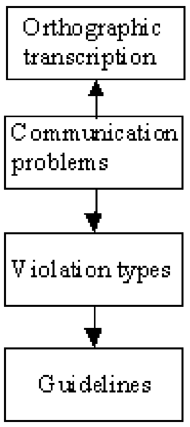

Figure 2. File organisation for a corpus annotated with respect to communication problems. An arrow from A to B means that elements in A refer to elements in B.

Such violation types are necessarily task-dependent as they refer to concrete problems found in dialogues with particular applications. An extensive collection of examples of communication problems, violation types, and references to the guidelines can be found in (Bernsen et al. 1998) and at http://www.disc2.dk/tools/codial/index.html.

The set of tags (elements and attributes) used by the communication problems coding scheme is small and simple, even if the three-component structure shown in Figure 2 is used. It is, however, a non-trivial task to identify communication problems and analyse them correctly to determine which guidelines they violate and how, i.e., which types of violation we are dealing with.

Guidelines may at times support one another, but at other times conflict when applied during actual interaction design. When guidelines conflict, the designers have to trade off different design options against one another, perhaps, for example, by giving the options a weighting of some kind depending upon the guideline(s) referred to. When designing a system introduction, for instance, developers may find that GG2 (don't say too much) conflicts with GG1 (say enough), SG4 (tell what the system can and cannot do), and SG5 (instruct on how to interact with the system). If the introduction is long and complex, even if all the points made are valid and important, users tend to get bored and inattentive. On the other hand, if the introduction is brief or even non-existent, important information may have been left out, increasing the likelihood of miscommunication during task performance.

During the detection and analysis of communication problems, an orthographic transcription of the dialogue is used. Often, however, the logfile will have to be inspected as well, cf. (Dybkjær et al. 1998b). In a few situations, it may even be necessary to have access to the sound files or to a phonetic transcription in order to determine the occurrence of an ambiguous utterance in the orthographic transcription (which would be clarified in the spoken version due to intonation). For example, some questions have the same form as statements, and only the information provided by the intonation will reveal whether it is one or the other.

3.1 Coding Modules

In the following, we present the coding modules for communication problems, violation types, and guidelines, cf. Figure 2. A coding module encapsulates the specification of a coding scheme. In terms of formal languages, a coding module can be seen as an abstract type or class specification, exposing its element declarations to the world. Slots for commenting on, e.g., coding purpose, coding level, and data sources are available, cf. (Dybkjær et al. 1998b).

Communication Problems Guidelines Coding Module

Name: Guidelines.
Coding purpose: Records the different generic and specific guidelines, the violation of which typically leads to communication problems in spoken human-machine dialogue.
Coding level: Communication problems.
Data sources: List of generic and specific guidelines for co-operative dialogue design.
Module references: None.
Markup declaration:

```
  ELEMENT aspect
  ELEMENT guideline
  ATTRIBUTES
    aspect: REFERENCE(this, as-
    pect)
    gricean: ENUM (yes|no)
    subsumed_by:
    REFERENCE(this, guideline)
    abbreviation: TEXT
```

Description: Two elements are used to annotate the guidelines. One is `aspect`. `aspect` is used to indicate a grouping of the guidelines. For example, the 24 guidelines in Figure 1 are divided into seven groups or aspects. The element `aspect` has no explicit attributes.

A second element is `guideline` which marks up a particular guideline. `guideline` has four attributes.

`aspect` is mandatory. It is a reference to the aspect to which the guideline belongs. The `aspect` indicated for a specific guideline must always equal the `aspect` indicated for the generic guideline by which it is subsumed.

`gricean` is mandatory for guidelines which are the same as Grice's maxims (Grice 1975). The `yes` value is used to indicate a maxim. For non-maxims `gricean` is optional. If indicated, the `no` value must be chosen. Using the value `yes` indicates whether a certain guideline is one of Grice's maxims.

`subsumed_by` should always be used for specific guidelines to indicate by which generic guideline it is subsumed. `subsumed_by` cannot be used for generic guidelines.

`abbreviation` is optional but recommended. It provides an abbreviated form of the guideline. It carries the essential meaning and may be easier to remember than the "canonical" expression of the guideline.

In fact, all elements also have a mandatory attribute `id` which is a unique identifier. If a tool is used the `id` should be generated automatically

Examples:

```
<aspect
id="1">Informativeness</aspect>
...
<guideline id="GG1" aspect="#1"
gricean="yes" abbreviation="Say
enough">Make your contribution
as informative as is required
(for the current purposes of the
exchange).</guideline>
<guideline id="SG1" aspect="#1"
subsumed_by="#GG1" abbrevia-
tion="State commitments explic-
itly">Be fully explicit in
communicating to users the com-
mitments they have
made.</guideline>
...
```

Coding procedure:
The guidelines for co-operative dialogue design are part of the coding module for communication problems defined below. However, they may also be re-used in other coding modules for communication problems. If a user, defining a new communication problems module, should want to build on a different set of guidelines it may well be that s/he can still reuse the coding module for guidelines defined here. Encoding a set of guidelines using the present coding module is not very complicated and the following procedure is recommended as sufficient:

- Encode by coder 1.
- Check by coder 2.

Creation notes:

Authors:	Hans Dybkjær and Laila Dybkjær.
Version:	1 (25 November 1998), 2 (19 June 1999).
Comments:	None.
Literature:	(Bernsen et al 1998).

Violation Types Coding Module

Name: Violation_types.
Coding purpose: Records the different ways in which generic and specific guidelines are violated in a given corpus, i.e. the types of problems found

in the corpus. The corpus is implicitly given by a communication problems coding file referring to the problem type coding file as well as to a transcription.
Coding level: Communication problems.
Data sources: List of types of violations of generic and specific guidelines for co-operative dialogue design. The list is generated during analysis of a corpus with respect to communication problems.
Module references: Module Guidelines.
Markup declaration:

```
  ELEMENT vtype
  ATTRIBUTES
    instance_of:
    REFERENCE(Guidelines,
    guideline)
    alternative_instances:
    REFERENCE(Guidelines,
    guideline+)
```

Description: Each description of a violation type is annotated by the element `vtype`. This element has two attributes.

The attribute `instance_of` is mandatory. `instance_of` is a reference to a particular guideline in a file which contains the guidelines for co-operative dialogue.

`alternative_instances` is optional. Guidelines overlap and in some cases the coder may be in doubt whether one or the other guideline was violated. The attribute `alternative_instances` allows the coder to express this doubt by letting him/her indicate one or more (this is what '+' means) other guidelines than the one referred to by `instance_of`.

The body of `vtype` contains the description of the actual type of violation.
Example:

```
<vtype id="SG4-1" in-
stance_of="Guidelines-
1999#SG4">Too little said on
what system can and cannot do:
BA often missing; time-table en-
quiries always missing.</vtype>
```

Coding procedure: Each communication problem is seen as a certain type of violation of a guideline. The violation types are highly task dependent. The file containing these types is built in parallel with the analysis and markup of communication problems. This file is very special in the sense that its contents, i.e. the text, as well as the markup are created at the same time and by the coder. The contents are textual descriptions of the violation types. We recommend to use the same coding procedure for violation types as for markup of communication problems since the two actions are tightly connected. As a minimum, the following procedure should be followed:

- Encode by coders 1 and 2.
- Check and merge codings (performed by coders 1 and 2 until consensus).

Creation notes:

Authors:	Hans Dybkjær and Laila Dybkjær.
Version:	1 (25 November 1998), 2 (19 June 1999).
Comments:	None.
Literature:	(Bernsen et al. 1998).

Communication Problems Coding Module

Name: Communication_problems.
Coding purpose: Records the different ways in which generic and specific guidelines are violated in a given corpus. The communication problems coding file refers to a problem type coding file as well as to a transcription.
Coding level: Communication problems.
Data sources: Dialogue corpora.
Module references: Module Basic_orthographic_transcription; Module Violation_types.
Markup declaration:

```
  ELEMENT comprob
  ATTRIBUTES
    vtype: REFERENCE (Viola-
      tion_types, vtype)
    wref: REFERENCE (Ba-
      sic_orthographic_transcri
      ption, (w,w)+)
    uref: REFERENCE (Ba-
      sic_orthographic_transcri
      ption, u+)
    caused_by: REFERENCE (this,
      comprob)
    temp: TEXT
    ELEMENT note
    ATTRIBUTES
    wref: REFERENCE (Ba-
      sic_orthographic_transcri
      ption, (w,w)+)
    uref: REFERENCE (Ba-
      sic_orthographic_transcri
      ption, u+)
```

Description: In order to annotate communication problems caused by inadequate systems design, we use the element comprob. It refers to some kind of violation of one of the guidelines listed in Figure 1. The comprob element may be used to mark up any part of the dialogue which caused the communication problem. Thus, it may be used to annotate one or more words, an entire utterance, or even several utterances in which a communication problem was detected. The comprob element has five attributes.

The attribute vtype is mandatory. vtype is a reference to a particular description of a guideline violation in a file which contains the different kinds of violations of the individual guidelines.

Either wref or uref must be indicated. Both these attributes refer to an orthographic transcription. wref delimits the word(s) which caused a communication problem, and uref refers to one or more entire utterances which caused a problem.

The attribute caused_by is optional. In some cases a communication problem in a dialogue will be caused by a problem which occurred earlier in that dialogue. caused_by is used to refer to a communication problem which was found elsewhere in the dialogue and which led to the present communication problem.

temp is an optional attribute. It indicates a temporary markup. It usually takes a few dialogues before the coder gets a good grasp of the types of guideline violations which tend to occur in the corpus and what caused them. Often logfile inspection will be needed to make an exact diagnosis. Moreover, some problems become easier to detect when comparing a few dialogues. Thus, temp is mainly for use during initial markup of a corpus but may also be used later if it is practical to make some temporary notes before making the final diagnosis. The vtype attribute overrides whatever communication problems the attribute temp indicates.

In the beginning of the analysis, the vtype attribute may be left open and the temp attribute filled in to describe the kind of guideline violation identified. Very soon, however, a file containing the violation types should be established and, in most cases, the temp comments can simply be moved to this file and possibly modified to provide a violation type description. Note that due to this and to the coding procedure requiring at least two coders, the violation type references in the vtype attribute are likely to eventually be re-classified.

The note element can be used anywhere in a corpus to comment on whatever the user wants to highlight. It refers to one or more words, or one or more utterances, in the same way as the comprob element. The body of the note element contains text.

Example:

The following example of communication problems markup assumes the snippet of a transcription from the SUNDIAL corpus below and refers to the example in the violation types coding module:

```
<u id="S1:7-1-sun"
who="S">flight information brit-
ish airways good day can I help
you</u>
<comprob id="3"
vtype="Sundial_problems#SG4-1"
uref="Sundial#S1:7-1-sun"/>
<note id="2" uref="Sundial#S1:7-
1-sun">The system provides too
little information about its ca-
pabilities and limitations. The
risk is that the user will be
misled and assume stronger or
weaker system capabilities than
are actually present. The intro-
duction suggests that users can
ask about anything to do with
British Airways flights. No cur-
rent system is likely to be able
to do that. Another interpreta-
tion of the system's introduc-
tion is that it is owned by
British Airways but can answer
any question about flights. The
former interpretation seems the
most natural one. So the sys-
tem's opening probably should
not be deemed ambiguous.</note>
```

Coding procedure: We recommend to use the same coding procedure for markup of communication problems as for violation types since the two actions are tightly connected. As a minimum, the following procedure should be followed:

- Encode by coders 1 and 2.
- Check and merge codings (performed by coders 1 and 2 until consensus).

Creation notes:

Authors:	Hans Dybkjær and Laila Dybkjær.
Version:	1 (25 November 1998), 2 (19 June 1999).
Comments:	For guidance on how to identify communication problems and for a collection of examples, the reader is recommended to look at http://www.disc2.dk/tools/codial/index.html.
Literature:	(Bernsen et al. 1998).

4 Application and Evaluation of the Coding Scheme

The guidelines have been applied to dialogues reflecting various task types, including flight ticket reservation, flight information, and train timetable information, and spanning development stages from early design to late evaluation. Also, the dialogue types have been different, including both system-directed and mixed-initiative dialogue.

We have tested intercoder agreement by applying the coding scheme to part of a corpus from the SUNDIAL project (Peckham 1993). The corpus comprises close to 100 early WOZ dialogues in which subjects seek time and route information on British Airways flights and sometimes on other flights as well. We selected 48 dialogues, such that each subject is represented with an approximately equal number of dialogues and each scenario (24 in total) is used in two dialogues. Two experts (A1 and A2) and one novice (A3) used the coding scheme. Three dialogues were used for training.

The two experts independently analysed 30 dialogues. Each detected violation was then discussed in detail and a typology of violations established. Violation types are task-dependent. The typology is useful for revising the dialogue model. The number of individual violations may support estimates of system performance and acceptability but is of little importance otherwise, as many violations are identical. In a corpus containing as many guideline violations as the SUNDIAL corpus, it is very time consuming, if not practically impossible, to find all the individual violations. It is also unnecessary, because what is needed for repairing the dialogue design are the *types* of guideline violations that occur. As shown in Figure 3, many *individual* violations were found by both experts (identities) but even more were found by either A1 or A2 (complementarity). However, all agreed violations could be classified under 24 different types. Of these, 15 were found by both experts whereas 9 types were found by either A1 or A2. Upon closer analysis, the cases belonging to 6 of the 9 complementary types turned out to be part of complex violations, i.e. utterances violating more than one guideline at the same time, which had been discovered by both experts. The remaining 3 types only covered 1 case each.

Having discussed and classified 30 dialogues, the experts analysed another 15 dialogues from the SUNDIAL corpus using the corpus-dependent typology established during the analysis of the first 30 dialogues. This facilitated dialogue annotation which could be reduced to references to a growing set of violation types. As shown in Figure 3, many more identical cases were found by the two experts in the last 15 dialogues. This is probably a result of their having discussed the findings in the first 30 dialogues. Slightly more type identities were found but also slightly more type complementarities.

	First 30 dialogues	Last 15 dialogues
Case identities (found by both experts)	81	92
Case complementarity (found by one expert)	133	41
Alternatives (different classifications)	7	3
Undecidable	1	0
Disagreements	21	2
Rejects	18	3
Type identities	15	17
Type complementarity	9	12

Figure 3. Results from the analysis of two sets of SUNDIAL dialogues by two experts in using the coding scheme.

However, all cases belonging to 8 of the 12 complementary types were part of complex violations that had been discovered by both experts. The remaining 4 types only covered one case each.

We also introduced a linguist to the coding scheme (A3). By way of introduction, A3 received the co-operativity guidelines (cf. Figure 1), a paper on their background and development, including examples of guideline violations, and a detailed coding scheme application walkthrough of three SUNDIAL dialogues. The complete analysis of one of these dialogues was given to him on paper. Having independently analysed a first set of 15 dialogues, A3 asked for, and had, a joint walk-through of one of those. A3 received no detailed written information on how to use the guidelines.

We analysed the correspondence between the findings of the two experts and those of the novice. Since the two experts had thoroughly discussed their findings after having analysed 30 of the 45 dialogues, thereby improving their performance on the last 15 dialogues, the following novice/expert comparison is based on the first 30 dialogues alone (cf. Figure 3, Column 2).

A3 found a total of 154 cases and 14 types, i.e. 80% of the average number of cases found by A1 and A2, and 72% of the average number of types found by A1 and A2. A3 found 10, or 42%, of the 24 types found by A1 and A2, and he found 4 new types. Three of these were part of complex violations that already had been observed by A1 and/or A2. The last type which covered only one case was not found by the two experts. Of the 154 cases found by A3, 26 cases were rejected, disagreed with, or considered undecidable by A1 and A2. This should be compared to an average of 20 such cases found by the two experts.

Taking into account that A3 never received any formal instructions on how to use the guidelines but had to generalise from examples, his performance would seem acceptable.

5 On-line versus Off-line Tagging

So far, tagging of communication problems using the NISLab coding scheme has always been done off-line at design time or evaluation time. Tagging has not been done on-line while a spoken dialogue system was running. The tagging has aimed to detect and diagnose problems in user-system interaction and, on this basis, propose improved dialogue model design. Tagging has not been viewed as a means to make on-line improvements of dialogue interaction.

Based on the communication problems coding described above, we are wondering whether, and in what sense, or faced with which kinds of phenomena, it might be possible to do on-line tagging-cum-repair of communication problems. It should be kept in mind that communication problems tagging, as described above, involves problem diagnosis followed by possible re-classification of the problem identified. This means that improvement may have to be done to any of the system's modules. Diagnosing a problem consists in deciding which module, if any, has to be corrected and how. For instance, it may be that the database has to be extended to better cover the domain, it may be that the output phrasing should be changed to avoid an obscure system utterance, or it may be that the dialogue manager needs better meta-communication abilities. It is hard to see how these modifications could be made on-line.

On the other hand, the spoken dialogue systems we build today actually do incorporate an increasing range of on-line mechanisms for bringing the dialogue back on track in case of, e.g., low recogniser confidence scores, parsing problems, user repair attempts, ambiguous user input, etc. In identifying and solving those problems, however, the system does not use tagging and coding schemes in any obvious sense of the term. Rather, some module detects the problem and the same or some other module fixes the problem.

References

Bernsen, N.O., Dybkjær, H. and Dybkjær, L.: Cooperativity in human-machine and human-human spoken dialogue. Discourse Processes, Vol. 21, No. 2, 1996, 213-236.

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

Dybkjær, L., Bernsen, N.O. and Dybkjær, H.: A methodology for diagnostic evaluation of spoken human-machine dialogue. International Journal of Human Computer Studies, 48, 1998a, 605-625.

Dybkjær, L., Bernsen, N.O., Dybkjær, H., McKelvie, D. and Mengel, A.: The MATE Markup Framework. MATE Deliverable D1.2, NISLab, 1998b.

Grice, P.: Logic and conversation. In Cole, P. and Morgan, J.L., Eds. Syntax and Semantics, Vol. 3, Speech Acts, New York, Academic Press, 1975, 41-58.

Mengel, A., Dybkjær, L., Garrido, J., Heid, U., Klein, M., Pirrelli, V., Poesio, M., Quazza, S., Schiffrin, A. and Soria, C.: MATE Dialogue Annotation Guidelines. MATE Deliverable D2.1, NISLab, 2000.

Peckham, J.: A new generation of spoken dialogue systems: Results and lessons from the SUNDIAL project. Proceedings of Eurospeech '93, Berlin, 1993, 33-40.