Proceedings of InSTIL/ICALL2004 – NLP and Speech Technologies in Advanced Language Learning Systems – Venice 17-19 June, 2004

Tutorial Dialogue in DiBEx

Manfred Klenner

Computational Linguistics, University of Zurich, Switzerland
klenner@cl.unizh.ch

Abstract

A **di**alogue-**b**ased **ex**planation facility for Intelligent CALL (ICALL) Systems is introduced. Our prototype system, DiBEx, uses meta reasoning to build up an explanation (error) tree, given a faulty user input. It relies on correct grammar sub theories, instead of explicit error taxonomies. The system enters in a tutorial dialogue with the student, where each explanation step is based on the principles of a single tutorial strategy. DiBEx focuses on the representation and utilization of grammatical knowledge to participate in such dialogues. Grammar theory is realized as an executable logical theory.

1. Introduction

DiBEx [1] is a prototype ICALL system that is able to participate in an explanatory tutorial dialogue. The focus is set on language generation tasks such as forming a sentence given a couple of words in base form or translating a sentence. Such an exercise could be: Given the words 'Mond aufgehen' (moon rise)`, form a full inflected sentence (in past perfect). The correct answer is: 'Der Mond ist aufgegangen' ('the moon has risen'). Here, among others, the learner must know that 'moon' ('Mond') is masculine in German (in contrast to e.g. Italian, where 'luna' is feminine). If the learner responds with 'Die Mond …', she probably is not aware of that (since 'die' followed by a singular noun must be feminine).

There is a wide range of possible error explanations, from specific ones ('moon is masculine in German') to abstract ones ('there is an agreement error [within the subject]'). But if a single explanation does not provide sufficient information to allow the learner to correct her input, a tutorial explanatory dialogue is needed.

Here is an example of such a top down dialogue:

system:	there is an agreement error
user:	where?
system:	at the subject
user:	why?
system:	article and noun do not agree in gender
user:	I can't fix the problem
system:	German 'Mond' is masculine
user:	I see, but what am I supposed to do?
system:	choose the corresponding masculine article
user:	what is it?
system:	'der'

DiBEx focuses on the representation and utilization of grammatical knowledge to participate in such dialogues. Grammar (sub) theory is realized as an executable logical theory (a Prolog program). A meta interpreter generates an explanation (proof) tree of the erroneous learner input. Explanations proceed top down, from clauses representing violated grammatical principles to leaf nodes that mark concrete violated facts. A crucial problem here is to hit the right level of abstraction: choosing the appropriate follow-up clause (e.g. implied descendents are not to be selected), but also selecting refering descriptions with the right granularity (e.g. definite or indefinite reference, word token or word class). Fig. 1 shows the overall design of DiBEx.

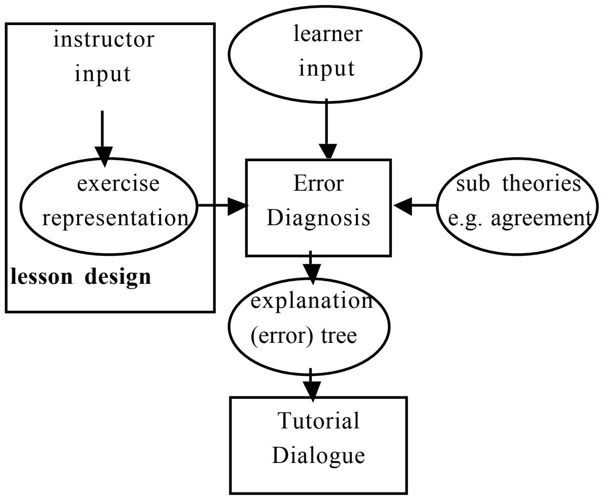

Figure 1. System architecture

A (LFG) parser is used to form an internal representation of the sentences that are provided by the instructor as part of the lesson design. Any faulty learner input is passed – together with the correct solution - to the error diagnosis component. In a first step, the user input is mapped to the correct solution whenever possible (e.g. deriving phrase boundaries from matching word sequences). Next, all grammar sub theories (e.g. agreement, word order, dominance) are invoked with the user input. If a sub theory fails, the correct solution, the user input and the sub theory are passed to a meta interpreter. As an result, an explanation error tree is generated and the tutorial dialogue is started.

In the next section, we describe the representation of grammatical knowledge. Then, the principles of error diagnosis done by DiBEx are introduced and finally, the tutorial component is discussed .

2. Grammatical knowledge

2.1 Grammar facts

In a language tutor, grammatical knowledge needs to be represented more explicitly than in standard NLP systems. For example, the principle of agreement in unification based grammars is captured by the unification of feature structures. So, the two feature structures [gender=mas] ('der') and [gender=mas] ('Mond') do unify. However, the fact that 'der' and 'Mond' should *agree* in gender is not explicitly represented. It is - so to say - built in. Such meta knowledge about grammar is necessary, if those parts of the domain knowledge that are violated by the user input should be identifiable. A possible, more explicit representation for that kind of knowledge is: *agree*(*noun*,*article*,*gender*). This means: The noun and the article must agree in gender.

This kind of representations is used by Menzel [2]. We build on his (slightly modified) representation format without using his reasoning scheme. For example, agreement information for 'der Mond' is represented by:

1) val(der1,[case,gen,num],[nom,mas,sg]).

2) val(mond1,[case,gen,num],[nom,mas,sg]).

That is: the value of 'der1' for case, gender, number is nominative, masculine, singular. Note that these facts are automatically generated from the instructor's input of the correct training sentence (by the LFG parser and a component that extracts the feature from the LFG f-structure). Other facts derived by the parser and grammatical background knowledge are:

3) val(der1,isa,det1).

4) val(mond1,isa,noun1).

5) val(subject1,has_part,[det1,noun1]).

6) val(sent1,has_part,[subject1,aux1,participle]).

Given a faulty user input ('die Mond …'), all available information from the lexicon is selected. E.g., for the word 'die':

7) val(die1,[case,gen,num],[nom,fem,sg]).

8) val(die1,[case,gen,num],[nom,mas,pl]).

Given these facts from the grammar and lexicon and given an error mapping ('die1' replaces 'der1'), some error messages (e.g. 'die' is wrong) or correction advice ('replace 'die' by 'der') could already be generated. However, to initiate an elaborated tutorial dialogue, additional knowledge is necessary. For dialogue steps like 'there is an agreement error' a theory of agreement is needed as well as a reasoning scheme that finds all theory parts that are violated by the user input.

2.2 Grammar rules

We represent grammatical rule knowledge as a logical theory in the programming language Prolog. Such a theory must serve two purposes: under a declarative point of view, it must be a valid description of the grammatical knowledge about a grammar sub theory (e.g. agreement), procedurally, it must be applicable to the user input in order to qualify the input as correct ('YES') or incorrect ('NO'). Fig. 2 represents a simplified theory for agreement.

```
agreement(Sent) :-
      subject_verb_agreement(Sent),        % omitted
      forall(phrase_of_sent(Sent,Phrase), % omitted
               phrase_agreement(Phrase)).

phrase_agreement(Phrase) :-
      val(Phrase,isa,noun_phrase),
      is_head(Phrase,Head),                  % omitted
      forall(non_head(Phrase,NonHead),
               cng_agree(NonHead,Head,[case,gen,num])).

cng_agree(NonHead,Head,CNG) :-
      val(Head,CNG,Vals1),
      val(NonHead,CNG,Vals2),
      forall(member(Feature,CNG),
               feature_agree(Feature,Vals1,Vals2)).

feature_agree(Feature,ValsHead,ValsNonHead) :-
      feature_val(ValsHead,Feature,Val1),
      feature_val(ValsNonHead,Feature,Val2),
      Val1=Val2.
```

Figure 2. A(partial) theory of agreement

Arguments with initial caps are Prolog variables (e.g. *Sent* in *agreement(Sent)*). The predicate *forall/2* applies its second argument, whenever its first argument is satisfiable. So line three and four from Fig. 2 can be paraphrased (in a declarative manner): for all phrases *Phrase* of sentence *Sent*, phrase agreement of *Phrase* holds.

Agreement, word order, dominance etc. are modelled in such a fashion in Prolog. The input of the learner must pass all those grammar sub theories. If the application of one sub theory fails, a meta interpreter is invoked, which derives an explanation (trace) of the faulty input.

3. Error diagnosis

In this section, we discuss the problem of error diagnosis which is a prerequisite for any error explanation. In general, parsing ill-formed sentences for error diagnosis is not trivial. The problem here is how to apply a correct and prescriptive grammar to an ungrammatical sentence. Without any further repair mechanisms, parsing simply fails. Solutions to that problem range from precompiled error grammars using mal-rules [3], to so-called anticipation free approaches like those described in [2,4]. Here, a constraint-based parsing formalism is used, and error diagnosis is accomplished through the relaxation of constraints.

In the DiBEx setting, the problem of error diagnosis is simpler, since the correct solution is known, i.e. is specified in advance by a human exercise creator.

The input of the student ('die Mond …') is automatically translated into grammatical facts by a lexicon lookup. E.g. the word 'die':

7) val(die1,[case,gen,num],[nom,fem,sg]).

8) val(die1,[case,gen,num],[nom,mas,pl]).

The hypothesis that (the incorrect) 'die Mond' is meant to be the *subject noun group* of the sentence 'die Mond ist aufgegangen' is crucial for the application of the grammar sub theory of agreement as specified in Fig. 2. We have defined a mapper that compares the incorrect input to the correct input. The incorrect input inherits from the correct input all non conflicting parts (e.g. that 'Mond' is part of subject1). To resolve conflicts, a model of heuristic 'error selection' is needed. We have just started to investigate such a theory. Consider the conflict at sentence position one: 'der' in the correct, 'die' in the incorrect solution. There are two options, given the grammatical facts in 7) and 8) above. 7) corresponds to a gender error (error source: learner thinks that 'Mond' is feminine as e.g. 'luna' in Italian is), 8) would indicate a number error (error source: learner thinks that 'Mond' is plural). Currently, we do not decide upon either possibility but just select one randomly. Note that there is a further diagnosis, namely that the learner thinks that 'die' is masculine (singular).

The mapper augments the user input in way that the sub theories are applicable to it. If one or more sub theories do fail to process the user input, error diagnosis is invoked with the user input, the correct solution and the failed sub theory.

Error diagnosis is realized as a meta interpreter. While traversing the sub theory with the correct solution, the incorrect solution is evaluated in parallel. Those predicates from the sub theory that failed upon the user input but succeeded upon the correct solution are marked. An explanation error tree is constructed from that.

This explanation is represented by a tree structure of the sub theory, where all variables are instantiated with the values from the user input and where, beginning from the root of the tree, a path down to a tree leaf indicates everything that is explicitly (a leaf node) or implicitly (intermediate nodes) violated by the user input. Let's have a look at the (slightly simplified) error tree given in Fig. 3.

The leaf node represents the violated fact, namely that the gender feature does not agree. The root node states that the grammar sub theory agreement does not hold for the input sentence. It is the task of the tutor to select the right level of abstraction. This includes the choice of an appropriate tree node to be verbalized with respect to the ongoing dialogue and the user model, but also the choice of how the arguments of the selected node should be referred to: 'die' could be referred to by 'die' or by 'the article' or even by 'a word' (although this might be too general in most circumstances). The tutor is introduced in the next section.

```
                false(agreement(main1))
                          |
         false(forall(phrase_of_sent(main1, A),
                  phrase_agreement(A)))
                          |
           false(phrase_agreement(subject1))
                          |
           false(forall(non_head(subject1, A),
            cng_agree(A, noun1),[case,gen,num]))
                          |
      false(cng_agree(det1, noun1,[case,gen,num]))
                          |
        false(forall(member(A, [case, gen, num]),
   feature_agree(A, [nom, mas, sg], [nom, fem, sg])))
                          |
           false(feature_agree(gen, mas, fem))
```

Figure 3. Explanation error tree

4. The DiBEx tutor

The tutor selects a node from the error tree that is assumed to be the best response (reaction) to a user question (or, initially, the faulty user input).

DiBEx currently uses the following tutorial strategy:

1. guide the user to identify the faulty fact in a top down manner, i.e. identify the grammatical principle violated by the input
2. provide sparse information, i.e. explain everything on an abstract level if possible (e.g. refer to a word by its word class).
3. introduce at each explanation step at least one piece of information that is new to the user
4. avoid becoming too vague (i.e. don't use more than one indefinite description)

Often such a top down strategy is called a Socratic (tutorial) Strategy, attributing this kind of dialogue to the Greek philosopher Socrates.

At the beginning of an tutorial dialogue, the root node of the tree is selected, i.e. *false(agreement(main1))*. According to tutorial principle 2, the tutor decides to suppress the referent *main1*, thus, 'There is an agreement error' is generated. Assume 'where?' as follow-up question. The argument of that predicate, *main1*, is not informative, since it represents the whole sentence. So DiBEx descends to the next node, a complex formula (*false(forall ...)*) that includes the quantifier *forall*. Such formulas (procedurally) define the grammatical principle represented by their immediate mother node. They are bad candidates for questions about concrete mistakes.

Descending, *false(phrase_agreement(subject1))* is reached. A failure of *phrase_agreement* is a failure of *agreement* and *subject1* is part of *main1*, thus *subject1* can be selected as an answer to the 'where' question: 'at the subject.'

We have implemented a preliminary model of tutorial coherence to operationalize this kind of reasoning. Among it's principles are rules that restrict the kind of reference to predicate arguments:

• given two arguments, they must be referred to at the same level:

*"'die' and the noun do not agree" is odd.

• reference to arguments must not be too general:

*'a function word is wrong' (given e.g. several function words)

• the reference to arguments may switch from abstract (word class) to specific (word form), but not the other way round. The sequence:

system: 'die is wrong'

user: 'why'

system: 'because the article does not agree with the noun'

is pragmatically ill-formed.

Let’s assume the next user question to be ’why?’, which means: why is there an agreement error at the subject? Skipping the next *forall* formula, we get:

false(cng_agree(det1, noun1,[case,gen,num])) and
false(feature_agree(gen, mas, fem).

According to tutorial principle 3, a piece of new information should be introduced at each step. If the user knows that the subject does not agree, she knows that it does not agree in case, number or gender (that's the definition of *phrase agreement*) But she does not know which one is violated. So: 'article and noun do not agree in gender' is an appropriate answer.

5. Related work

Besides the already discussed work from the ICALL community, work done in the field of intelligent tutorial systems (ITS) also is of interest to DiBEx. In [5], a geometrical tutor is described. Here, the student is requested to state (in natural language) an explanation of a previously learned geometrical theorem. The task of the tutor is to evaluate and criticize the student explanation. The ultimate goal is to help the student to form a complete and correct statement about a geometrical rule.

Johanna Moore's PEA (Program Enhanced Advisor) [6] is a tutor that helps students to improve their programming skills in Lisp. An expert systems called EES is used to generate a so-called development history that fixes the reasons for correction advices. Follow-up why questions use the development history to provide more precise reasons for the advice. PEA uses Rhetorical Structure Theory to integrate various knowledge resources (domain knowledge, discourse history, a speech act and user model) into a single model component.

6. Conclusion and outlook

We argued in this paper that an explicit representation of grammatical knowledge can be used for the verification of user input as well as for error diagnosis and error explanation. An error tree generated by a meta interpreter serves as the starting point for a tutorial dialogue that (currently) is based on a single Socratic tutorial strategy: provide sparse error descriptions, give hints of why something is wrong. Each node in the error tree can be used as a more or less abstract error explanation. A why question posed by the student in response to such an explanation can be resolved by ascending the tree to a subordinated (negated) node, leading to a more precise error description.

Future work will be concerned with the development of a model of heuristic 'error selection', as well as with a model of 'explanational coherence' for tutorial dialogue.

References

[1] Klenner, M. and Visser H. (2003). What exactly is wrong and why? Tutorial Dialogue for Intelligent CALL Systems. In: K-U. Carstensen (ed.)(2003), *Learning and teaching (in) Computational Linguistics*, Linguistik Online 17 (5/03): 57-80.

[2] Menzel, W. (1992): *Modellbasierte Fehlerdiagnose in Sprachlehrsystemen.* Tübingen.

[3] McCoy, K.. F., Pennington, C. A. and Suri, L. Z. (1996): "English error correction: A syntactic user model based on principled 'mal-rule' scoring". In: *Proc. of the Fifth International Conference on User Modeling*. Newton, MA: 59-66.

[4] Menzel, W. (1988): "Diagnosing grammatical faults - a deep modelled approach". In: O'Shea, Tim/Sgurev, Vasil (eds.): *Artificial Intelligence III - Methodology, Systems, Applications*. Amsterdam: 319-326.

[5] Aleven, V., Popescu, O. and Koedinger, K. (2001): "Towards tutorial dialog to support self-explanation: Adding natural language understanding to a cognitive tutor". In: Moore, J. D./Redfield, C. L./Johnson W. L. (eds.): *Artificial Intelligence in Education. Proc. of AI-ED 2001*: Amsterdam 246-255.

[6] Moore, J. D. (1995): *Participating in explanatory dialogues*. Cambridge, MA.