Act-Topic Patterns for Automatically Checking Dialogue Models

Hans Dybkjær and Laila Dybkjær

Prolog Development Center A/S (PDC)
H. J. Holst Vej 3C-5C
2605 Brøndby, Denmark
dybkjaer@pdc.dk

Natural Interactive Systems Laboratory
University of Southern Denmark
Campusvej 55, 5230 Odense M, Denmark
laila@nis.sdu.dk

Abstract

When dialogue models are evaluated today, this is normally done by using some evaluation method to collect data, often involving users interacting with the system model, and then subsequently analysing the collected data. We present a tool called DialogDesigner that enables automatic evaluation performed directly on the dialogue model and that does not require any data collection first. DialogDesigner is a tool in support of rapid design and evaluation of dialogue models. The first version was developed in 2005 and enabled developers to create an electronic dialogue model, get various graphical views of the model, run a Wizard-of-Oz (WOZ) simulation session, and extract different presentations in HTML. The second version includes extensions in terms of support for automatic dialogue model evaluation. Various aspects of dialogue model well-formedness can be automatically checked. Some of the automatic analyses simply perform checks based on the state and transition structure of the dialogue model while the core part are based on act-topic annotation of prompts and transitions in the dialogue model and specification of act-topic patterns. This paper focuses on the version 2 extensions.

1. Introduction

We present a new approach to evaluating spoken dialogue systems (SDSs) where early, electronic dialogue models can be evaluated automatically, rather than only via resource-demanding corpus collection with users and subsequent evaluation involving considerable human effort.

Dialogue model design errors should preferably be caught early in the software development process. Evaluation methods like Wizard-of-Oz (WOZ), walk-throughs and guidelines support detection of design problems. Also, tools exist that support rapid design and evaluation of dialogue models. We have ourselves developed DialogDesigner version 1, see [Dybkjær and Dybkjær 2005] and www.spokendialogue.dk/DialogDesigner, which has been used in commercial development processes since early spring 2005. However, this and other tools as well as methods offer little or no support for automatic evaluation of dialogue models.

Automatic evaluation requires some kind of formalisation of dialogue evaluation. We have earlier discussed act-topic patterns as a formalised way in which to express well-formedness of task-oriented dialogue in SDSs [Dybkjær and Dybkjær 2004]. We are now building on this work while extending DialogDesigner. Our aim is to enable DialogDesigner to automatically:

1. check if the dialogue model is well-formed in different respects, including whether it satisfies well-formed act-topic patterns, and
2. identify certain types of interaction problems in logs from WOZ experiments or from the implemented system.

We concentrate on point 1 in this paper corresponding to the extensions made in DialogDesigner version 2.

In the following Section 2 briefly presents DialogDesigner. Section 3 explains how to create an electronic dialogue model in DialogDesigner. Section 4 focuses on act-topic annotation, while Section 5 presents the various automatic analyses offered by the tool and provides illustrations of act topic patterns and their use. Section 6 concludes the paper.

2. DialogDesigner

Harris [2005] stresses the importance of having an electronic dialogue model because that enables us to add and exploit various kinds of support for development and evaluation.

DialogueDesigner version 1 was built in 2005 to support the creation of electronic dialogue models of task-oriented SDSs. The tool is implemented in C# and runs on a Windows platform. The basis in DialogDesigner is the design window where one can enter and browse a dialogue model, including prompts, conditions, and state transitions, cf. Figure 1. Once a dialogue model has been entered there are various presentation possibilities. The user may see graphical views of the dialogue model, or run a Wizard-of-Oz (WOZ) simulation with users or – using the same tool – make a walkthrough of the dialogue model. The simulation or walkthrough is logged and can be saved for later analysis and commenting. The simulation log can also be used normatively to generate test scripts for use in a systematic functionality test of the implemented system. Finally, the user may extract HTML versions of the entire dialogue and of prompt and phrase lists. The creation of a dialogue model with DialogDesigner version 1 and the mentioned presentation possibilities are thoroughly described in [Dybkjær and Dybkjær 2005].

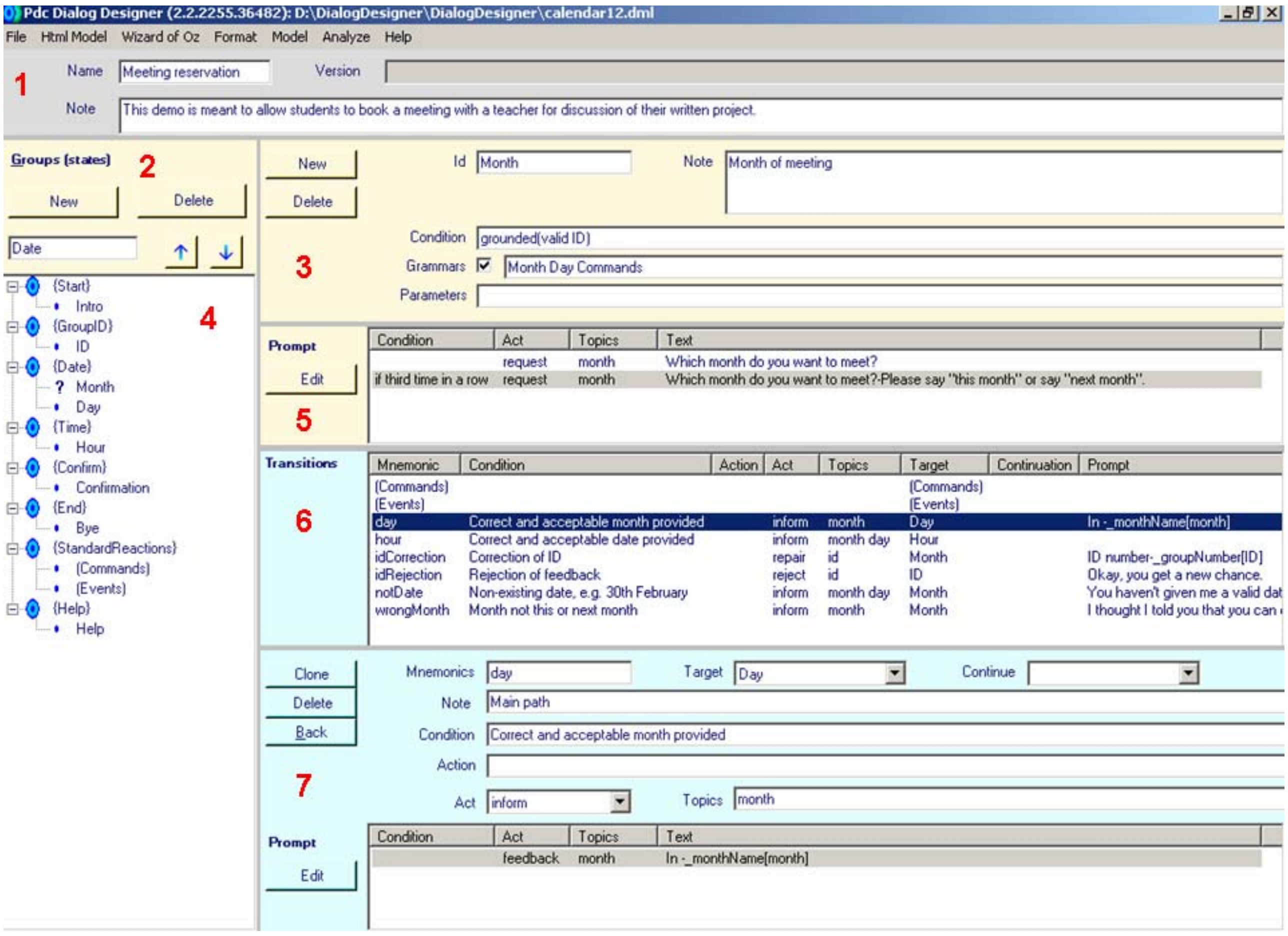
Version 2 of DialogDesigner includes two important extensions:

1. It enables act-topic annotation of prompts and transitions in the dialogue model at design time.
2. It supports various kinds of automatic analyses of the dialogue model most of which are based on the act-topic annotation of prompts and transitions in the dialogue model.

3. Entering a Dialogue Model in DialogDesigner

In the following we briefly explain the design and prompt windows of DialogDesigner and how to enter a dialogue model, including how to mark up acts and topics. The explanation refers to the red numbers in Figures 1 (design window) and 2 (prompt window), To make the references clearer we shall use D in front of numbers referring to the design window and P in front of numbers referring to the prompt window

The pane at D1 is for administrative information. At D2 groups are added and named. A group consists of one or more dialogue states which conceptually belong together and are described by the group. New states are added at D3. If there are certain conditions for entering a state the condition field should be filled. A condition on the state “hour” could e.g. be that you must have agreed on a date before you can negotiate a particular time slot for a meeting. At D4 the hierarchy of defined groups and states is shown.



Name: Meeting reservation
Version:
Note: This demo is meant to allow students to book a meeting with a teacher for discussion of their written project.

Date

Id: Month
Note: Month of meeting
Condition: grounded(valid ID)
Grammars: ✓ Month Day Commands
Parameters:

Condition	Act	Topics	Text
	request	month	Which month do you want to meet?
if third time in a row	request	month	Which month do you want to meet?-Please say "this month" or say "next month".

Mnemonic	Condition	Action	Act	Topics	Target	Continuation	Prompt
{Commands}					{Commands}		
{Events}					{Events}		
day	Correct and acceptable month provided		inform	month	Day		In -_monthName[month]
hour	Correct and acceptable date provided		inform	month day	Hour		
idCorrection	Correction of ID		repair	id	Month		ID number-_groupNumber[ID]
idRejection	Rejection of feedback		reject	id	ID		Okay, you get a new chance.
notDate	Non-existing date, e.g. 30th February		inform	month day	Month		You haven't given me a valid dat
wrongMonth	Month not this or next month		inform	month	Month		I thought I told you that you can

Mnemonics: day
Target: Day
Continue:
Note: Main path
Condition: Correct and acceptable month provided
Action:
Act: inform
Topics: month

Condition	Act	Topics	Text
	feedback	month	In -_monthName[month]

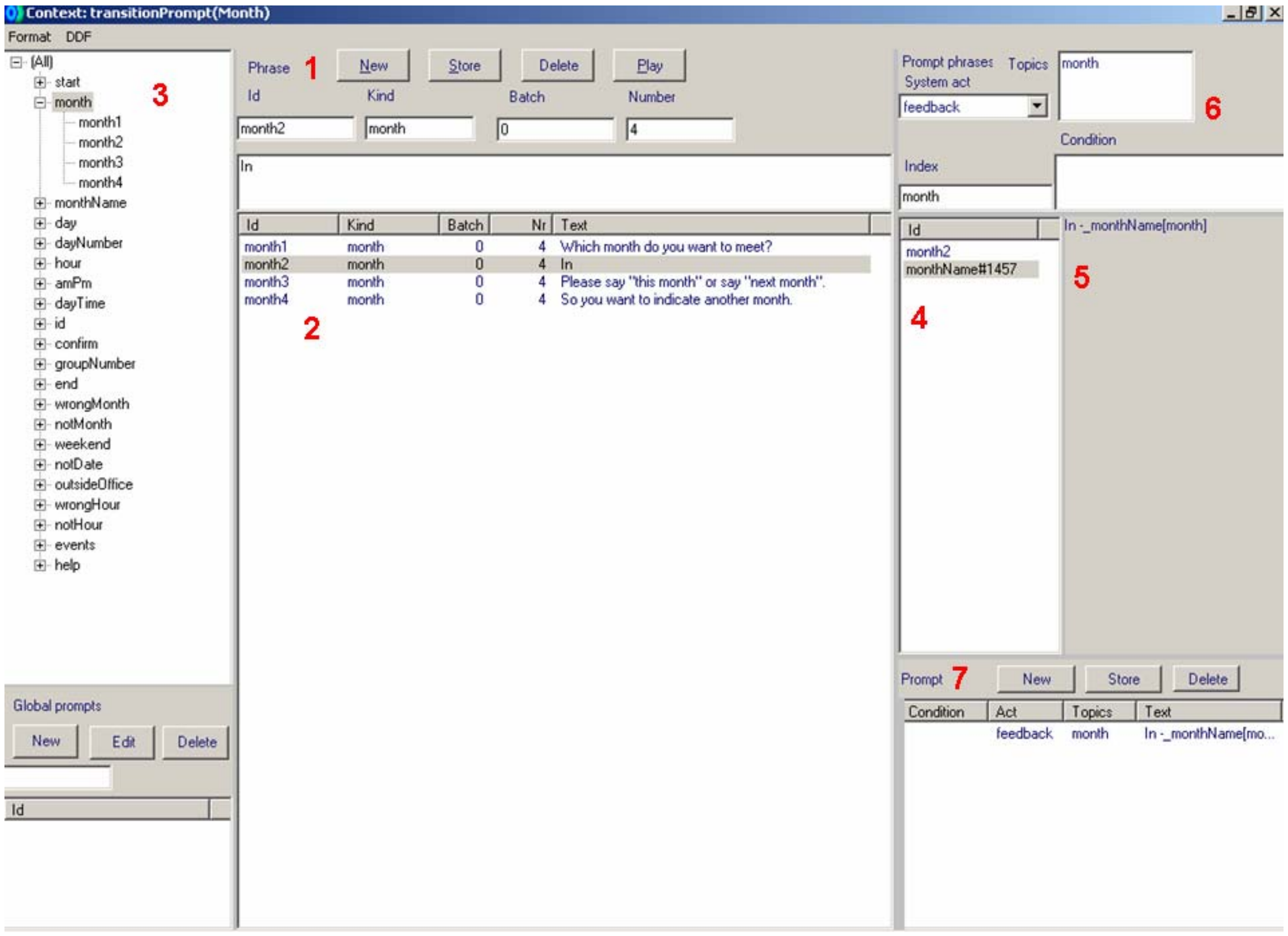
Figure 1. The designer window. Red numbers are used for reference in the text.

Prompts are entered at D5 via the “Edit” button which opens a new window, cf. Figure 2. In this window new phrases are entered at P1 and shown at P2. The “kind” is used for grouping phrases together. At P3 the defined kinds are shown. Each kind subsumes one or more ids. Composition of a new prompt is initiated at P7. A prompt is then composed by double clicking the relevant phrase at P2. The id of the selected phrase will be shown at P4 and the actual phrase is shown at P5. Prompts may be composed from several phrases. Each new selected phrase will be appended to the previous one(s) at P4 and P5. If, instead of a phrase a kind name at P3 is selected, a variable will be added to the prompt that may take several values, e.g. any of the twelve month names as illustrated in Figure 2. For each prompt phrase it is possible to indicate a system act and one or more topics at P6. Indication of acts and topics for each prompt is mandatory for most of the automatic analyses to work, cf. Section 4. A state may have several conditioned output prompts, cf. D5 in Figure 1.

When the prompt window is closed the entered prompt is shown at D5 in Figure 1 along with any specified condition, act and topic(s).

For each state there will very often be several possible transitions. These are entered at D7 and displayed at D6 in Figure 1. Among the information that may be entered for each transition is the act and topic(s) of the user’s input which are a precondition for choosing this transition. The prompt field at the bottom of the window at D7 works in the same way as the prompt field at D5, including indication of acts and topics. It is used to indicate e.g. feedback given when moving from the current state to the target state.

Formally we may view the dialogue model as a graph where nodes are states with prompt sets, and edges are transitions that are pairs of a user input (or exceptional events like database failure and timeouts) and a system prompt set (think of this as an explicit feedback point). Both state and transition prompts are annotated with system act-topics, while both user inputs and exceptional events are annotated with user act-topics. The graph is conditional in several places: Every state, every prompt, and every transition are guarded by a condition.



Phrase — Id: month2; Kind: month; Batch: 0; Number: 4

In

Id	Kind	Batch	Nr	Text
month1	month	0	4	Which month do you want to meet?
month2	month	0	4	In
month3	month	0	4	Please say "this month" or say "next month".
month4	month	0	4	So you want to indicate another month.

System act: feedback; Topics: month; Index: month

Id
month2
monthName#1457

In -_monthName[month]

Condition	Act	Topics	Text
	feedback	month	In -_monthName[mo...

Figure 2. The prompt window. Red numbers are used for reference in the text.

4. Act-Topic Annotation

As explained above, each system prompt and each user input utterance is described in terms of a dialogue act and one or more topics addressed. We have previously [Dybkjær and Dybkjær 2004] proposed to annotate transcribed dialogues with acts and topics to automatically identify transaction success and dialogue smoothness. Hastie et al. [2002] use annotated system utterances, but disregard user utterances, as a basis for automatic annotation of task-completion which is not necessarily the same as task success. Once we neglect user utterances, the dialogues by definition follow the dialogue model.

The approach we take in DialogDesigner builds on ideas from [Dybkjær and Dybkjær 2004] but does not use transcriptions or recognised dialogues as its basis. Rather we use the dialogue model itself as point of departure. To exploit the automatic analysis functionality in the extended DialogDesigner each prompt and each transition in the dialogue model must be annotated with speech acts and topics.

Speech acts are frequently used in SDS research. Nevertheless there is far from any agreement on a standard set of speech acts. Some use fairly few acts while others use many, and the set of speech acts used may vary considerably even when the number of speech acts is more or less the same. We believe that it is generally acknowledged that for SDS research there is normally a need for something more fine-grained than e.g. Searle's five speech acts [Searle 1969, 1979]. We also believe that some degree of reuse of speech acts across applications may be possible.

Speech act	Explanation
accept	Speaker A accepts an offer or a check made by speaker B.
check	Speaker A checks if the understanding of what was just heard is correct.
clarify	Speaker A asks for disambiguation.
feedback	Speaker A provides feedback on what speaker B just said.
hangup	Speaker A leaves the dialogue.
inform	Speaker A provides information to speaker B.
offer	Speaker A offers information to speaker B.
other	Speaker A makes unclear or null action.
pause	No ínput was recorded.
reject	Speaker A rejects an offer, information or feedback from speaker B.
repair	Speaker A corrects information, feedback or other action from speaker B.
repeat	Speaker A asks for repetition of what speaker B said.
request	Speaker A asks for information from speaker B.
select	Speaker A selects a topic offered by speaker B.

Figure 3. Dialogue acts in DialogDesigner. Note that speaker A may be the system as well as the user. The same is the case for speaker B.

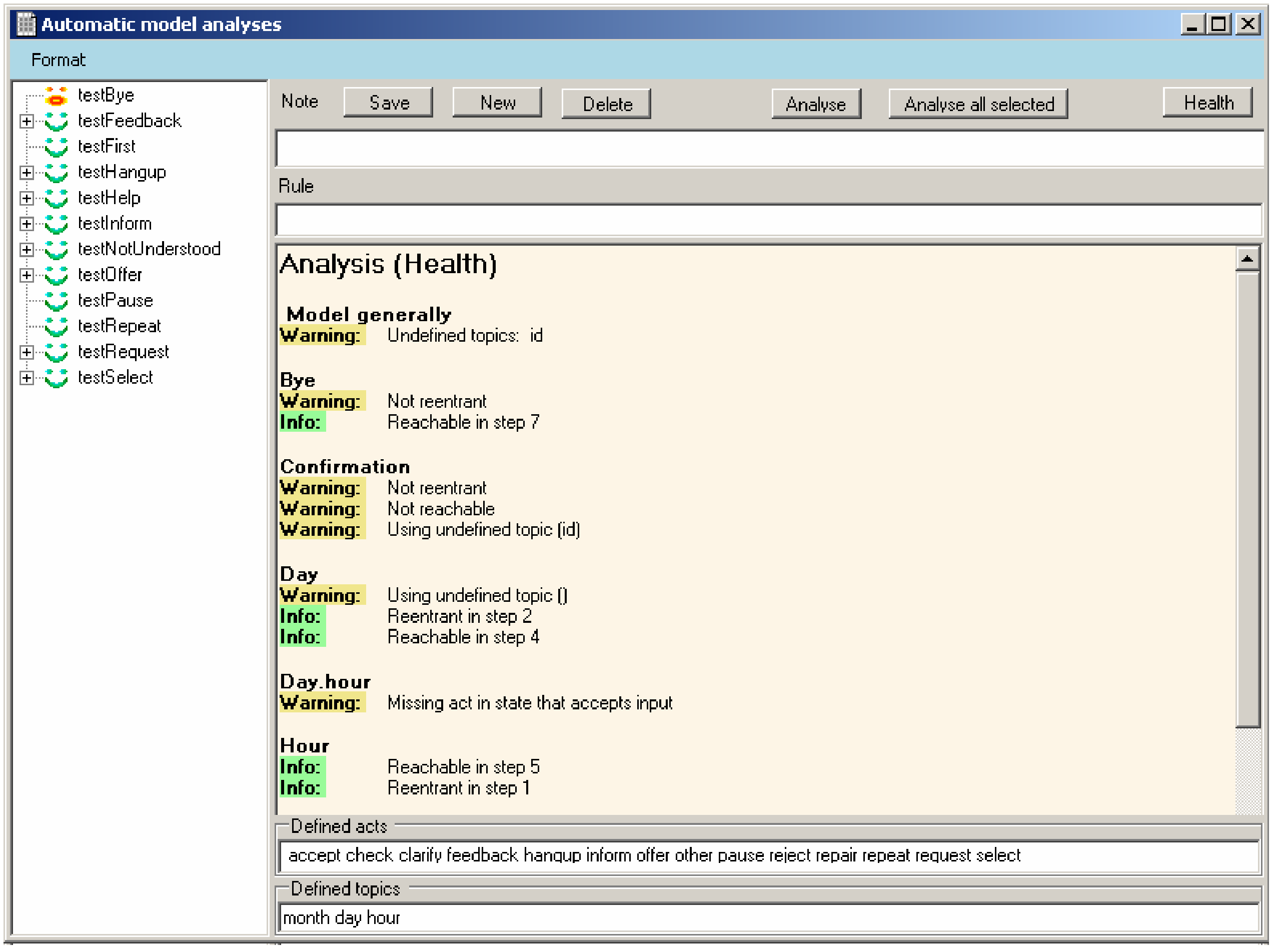
However, no standard set of speech acts exists and hardly can exist. What is an appropriate set of speech acts is highly dependent on the sort of analysis one wants to perform. Therefore, we need to allow changes to the default set of speech acts provided by DialogDesigner. DialogDesigner version 2 comes by default with the speech acts shown and explained in Figure 3, but may be configured to other sets.

Topics are highly domain and task dependent. Therefore the user of DialogDesigner has to define his own set of topics for any dialogue model.

5. Automatic Analyses

The “Analyze” menu point (Figure 1) is an addition in DialogDesigner version 2. When “Analyze” is selected a new window will open that provides access to a number of automatic analyses, cf. Figure 4.

The window in Figure 4 supports a “Health” check of the dialogue model via the Health button, and it supports the specification of rule patterns and subsequent analysis of whether the dialogue model conforms to the patterns, see Figure 5.



testBye
testFeedback
testFirst
testHangup
testHelp
testInform
testNotUnderstood
testOffer
testPause
testRepeat
testRequest
testSelect

Analysis (Health)

Model generally
Warning: Undefined topics: id

Bye
Warning: Not reentrant
Info: Reachable in step 7

Confirmation
Warning: Not reentrant
Warning: Not reachable
Warning: Using undefined topic (id)

Day
Warning: Using undefined topic ()
Info: Reentrant in step 2
Info: Reachable in step 4

Day.hour
Warning: Missing act in state that accepts input

Hour
Info: Reachable in step 5
Info: Reentrant in step 1

Defined acts
accept check clarify feedback hangup inform offer other pause reject repair repeat request select

Defined topics
month day hour

Figure 4. Results from an automatic analysis of whether it is possible to (1) reach each state in the dialogue model, (2) return to each state in the dialogue model, and (3) if there are prompts and transitions with no dialogue act indicated.

5.1. Health Analysis

“Health” currently performs four kinds of analysis on the dialogue model as described in the following.

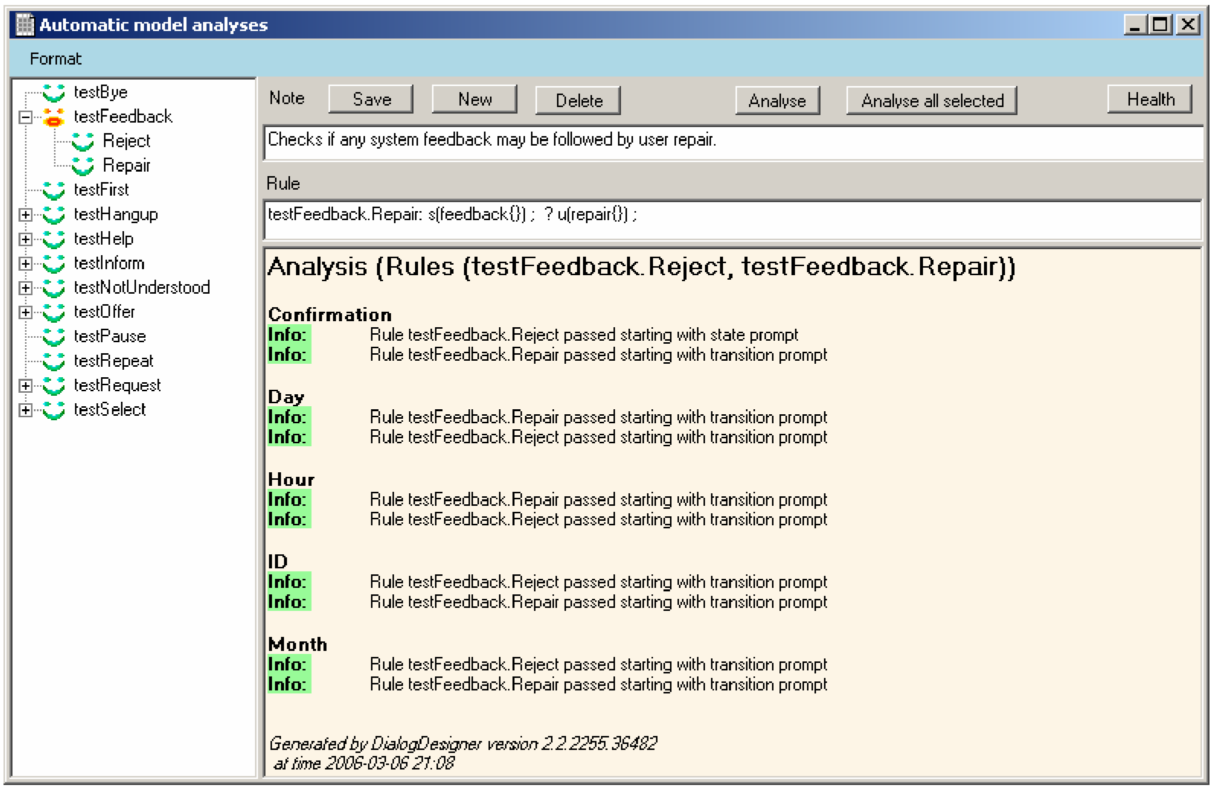
One analysis concerns reachability. It is a check of whether it is at all possible from the initial state to reach any other state defined in the dialogue model. For each state the output is either a warning that the state cannot be reached or information about how many steps it as a minimum takes to reach the state, cf. Figure 4.

The second analysis is a check of whether one can get back to each state in a finite number of steps, i.e. is the state “re-entrant”. To prevent self-transitions from making a state re-entrant, the analysis of whether a given state is re-entrant only looks at whether there are transitions back to the state in question from any of the subsequent states. The analysis either gives a warning that a state is not re-entrant or it informs about how many steps it as a minimum takes to get back to the state, cf. Figure 4.

Note that the two “Health” analyses just described are not based on act-topic annotation but they do provide information regarding the well-formedness of the analysed dialogue model.

The third analysis checks each prompt and each transition to see if a speech act has been indicated and if it is one of the defined acts. Whenever a prompt or a transition without a speech act is found, the analysis issues a warning. The speech acts are those listed near the bottom of Figure 4. The list of defined acts appears automatically.

The fourth analysis draws on a list of developer defined topics which are written in the field at the bottom of Figure 4. The analysis checks if topics are used that are not in the list and gives a warning about the ones it finds.



Checks if any system feedback may be followed by user repair.

testFeedback.Repair: s(feedback{}) ; ? u(repair{}) ;

Analysis (Rules (testFeedback.Reject, testFeedback.Repair))

Confirmation
Info: Rule testFeedback.Reject passed starting with state prompt
Info: Rule testFeedback.Repair passed starting with transition prompt

Day
Info: Rule testFeedback.Repair passed starting with transition prompt
Info: Rule testFeedback.Reject passed starting with transition prompt

Hour
Info: Rule testFeedback.Repair passed starting with transition prompt
Info: Rule testFeedback.Reject passed starting with transition prompt

ID
Info: Rule testFeedback.Reject passed starting with transition prompt
Info: Rule testFeedback.Repair passed starting with transition prompt

Month
Info: Rule testFeedback.Reject passed starting with transition prompt
Info: Rule testFeedback.Repair passed starting with transition prompt

Generated by DialogDesigner version 2.2.2255.36482
at time 2006-03-06 21:08

Figure 5. Analysis results from applying the selected rules to the dialogue model.

5.2. Analysis Using Act-Topic Patterns

Figure 5 shows the same window as Figure 4 but focus is now on act-topic patterns, also called *rule patterns*. They are entered in the "Rule" field. An explanation of what the rule does may be entered in the "Note" field. The set of defined rules is listed in the tree structure to the left.

The "Analyse" button analyses the current rule in the Rule field. If a rule or group of rules to the left is selected and you subsequently click "Analyse selected", the analysis result will be shown in the large pane at the bottom to the right. Figure 5 shows the results of analysing two different rule patterns in one go, i.e. those subsumed under testFeedback.

5.2.1. Writing Act-Topic Patterns

Rules are simply act-topics sequences written on the following form:

RULE = NAME: CONDITION '?' TURN*
CONDITION = ['^'] TURN*
TURN = WHO '(' ACTTOPICS+ ')' ';'
WHO = 's' | 'u' | '_'
ACTTOPICS = ACT '{' TOPIC* '}'
ACT = '_' | ACTNAME

An example is

```
testFeedback.Repair:
  s(feedback{}) ;  ? u(repair{}) ;
```

The "s" (system) and "u" (user) are used to indicate who performs which act. Feedback and repair are speech acts. The "{}" indicates any topic(s), i.e. in the example we don't care which topic(s) the user and the system are addressing. The example rule will cause the analysis to check if user repair is possible whenever there is system feedback.

Another example is

```
testHelp:
  s(_{}) ; u(request{help}) ; ?
  s(inform{help});
```

The "_" indicates any speech act. Thus the rule means that whenever the system has performed an act and the model allows the user to make a request for help, it must be possible for the system to provide help.

We could make the following very similar rule instead in which only the first speech act is the condition:

```
testHelp:
  s(_{}) ; ? u(request{help}) ;
  s(inform{help});
```

The meaning of this rule is that whenever the system has performed some speech act it must be possible for the user to ask for help and get help.

The rule matches are existential only: The analysis will succeed for a state if just one match with the rule patterns is found. The analysis does not check if there are several matches for the same state. Also, the analysis is an abstraction in the sense that it relies on the act-topic annotation without computing the condition fields of the model. In principle the act-topic annotation must be coherent with the conditions. However, in practice it means that although the analysis shows that a path is possible, the actual runtime conditions may turn out to not allow the path.

5.2.2. Using Act-Topic Patterns

By applying the specified rule patterns and the act-topic annotated dialogue model we can check if the dialogue model is well-formed in various ways, i.e. if it follows the specified rule patterns. We have specified act-topic patterns to perform the following analyses:

- Universals: In any input state, universals such as repetition, help, and goodbye should be included.

- Events: NothingUnderstood, Timeout, Hang-Up etc. must be handled everywhere.
- Feedback: The user should be able to reject or repair feedback from the system. Moreover, it may be desirable that user inform or select acts are followed by feedback from the system. We also check this.
- Common act sequences: There are several, e.g.:
    - If the system makes an offer, it must be possible for the user to reject the offer or to accept or select something from the offer.
    - If the user has selected an offer it must be possible for the system to provide information.
    - If the system requests information it must be possible for the user to provide information.
- Topic reactions: Requests concerning a topic T must be followed by a response concerning T.

We have applied the rules to two demo dialogue models and to a dialogue model for a commercial system. One demo was iteratively developed for testing and experimenting with DialogDesigner version 2. It is a small calendar application which allows students to book a time slot in the teacher's calendar for discussion of their project. A second demo application concerns pizza ordering. It was originally developed to experiment with DialogDesigner version 1. The original dialogue model has subsequently been annotated with acts and topics. The dialogue model used in a commercial traffic information system was originally designed using DialogDesigner version 1. This dialogue model has also subsequently been act-topic annotated.

We have used the same set of dialogue acts in all three cases but of course the topics are very different. Rules can easily be copied from on model to another. We did so but only used those relevant to the respective models. For example, the calendar application does not include any "offer" speech acts.

The calendar application has been iteratively analysed and the various analyses have helped pointing out weaknesses and omissions in the dialogue model which could then immediately be corrected. The analyses revealed several issues for consideration in the pizza demo. This was expected since the model is at an early stage but it is nevertheless helpful to get a list of problems.

The traffic information model is relatively small and quite mature in that it has been tested and running in production for over a year. Nevertheless the analysis revealed a couple of empty transitions which should just be deleted, and more important, a state where the previous value could not be repaired, essentially meaning that a whole roundtrip will be needed in this dialogue if a wrong choice is taken.

6. Conclusion and Future Work

We have presented DialogDesigner which is a tool in support of rapid design and evaluation of dialogue models. Focus has been on the version 2 extensions in terms of support for automatic dialogue model evaluation. Various aspects of dialogue model well-formedness can be automatically checked. Two of the presented analyses perform checks based on the state and transition structure of the dialogue model while all other described analyses are based on act-topic annotation of prompts and transitions in the dialogue model and specification of act-topic patterns.

We are planning to experiment with several additional extensions of DialogDesigner. We describe the primary ones below.

All rules must for the moment be forward looking, i.e. they can only check if some speech act occur after the condition. We are planning to also allow backward looking rules. These may be convenient in a number of cases. Suppose, e.g. that you have a system which should offer discount only if the user orders a return ticket. Then you may have a rule which checks if, provided you find a system offer for discount, there has previously been a user request or select speech act for a return ticket. Note that this backwards analysis would mean "for act A, *all* paths leading to A must contain act B". This is in contrast with forward looking rules that are existential statements only.

A second extension we are planning involves making the rule notation more expressive, e.g. by introducing regular expression operators, such as '*', '+', and '[ ]'.

A third extension concerns analysis results. Given a positive analysis result it may be desirable to be able to view the actual sequence of prompts and transitions that satisfied the rule pattern.

Fourthly we plan to allow analysis of whether one or more required paths through the dialogue model exist. You may think of such paths as reflecting smooth traversals of the required main system functionalities, e.g. reservation of a one-way ticket and of a return ticket.

A fifth improvement would be to make a closer approximation to the runtime situation by improving the condition handling. However, due to the insolubility of the halting problem, we can never make an exact analysis.

A sixth extension relates to the log files from Wizard of Oz sessions or from interaction with the implemented system. Based on the act-topic annotation it is possible to automatically identify locations in the logs where the interaction is not smooth in the sense that e.g. repair acts occur. This enables the developer to quickly look at the identified locations to see if there is an interaction problem which requires changes to the dialogue model.

Finally, to repeat our main message from earlier: *Having an electronic model is great for designing, discussing, presenting and probing the model during design, development, test, and evaluation.* When the model is electronic, all sorts of analyses are easily added.

References

Dybkjær, H. and Dybkjær, L.: From Acts and Topics to Transactions and Dialogue Smoothness. Proceedings of LREC2004, Vol. V, Lisbon, Portugal, 2004, 1691-1694.

Dybkjær, H. and Dybkjær, L.: DialogDesigner – A Tool for Rapid System Design and Evaluation. Proceedings of Sixth SIGdial Workshop on Discourse and Dialogue, Lisbon, Portugal, 2005, 227-231.

Harris, R.: Voice Interaction Design. Morgan Kaufmann Publishers, 2005.

Hastie, H. W., Prasad, R. and Walker, M.: Automatic evaluation: Using a DATE Dialogue Act Tagger for User Satisfaction and Task Completion Prediction. Proceedings of LREC2002, 641-648.

Searle, J. R.: Speech Acts. An Essay in the Philosophy of Language. Cambridge University Press, 1969.

Searle, J. R.: Expression and Meaning. Studies in the Theory of Speech Acts. New York, Cambridge University Press, 1979.