Some Notes on the Complexity of Dialogues *

Jan Alexandersson
DFKI GmbH, Stuhlsatzenhausweg 3,
D-66123 Saarbrücken,
Germany
janal@dfki.de

Paul Heisterkamp
DaimlerChrysler AG
Wilhelm-Runge-Str. 11
D-89081 Ulm, Germany
paul.heisterkamp@DaimlerChrysler.com

Abstract

The purpose of this paper is twofold. First, we describe some complexity aspects of spoken dialogue. It is shown that, given the internal setting of our dialogue system, it is impossible to test even a small percentage of the theoretically possible utterances in a reasonable amount of time. An even smaller part of possible dialogues can thus be tested. Second, an approach for early testing of the *dialogue manager* of a dialogue system, without the complete system being put together, is described.

1 Introduction

On the one hand, it is important for the developers of a dialogue system that the system is robust (i.e., it does not fail or loop), easy to use and is efficient. On the other hand, the testing of a dialogue system is cumbersome and expensive. Factors like the effectiveness and naturalness of the system, as well as robustness are problematic to evaluate. While test suites for analysis components have been around for a while, their counterparts for *dialogue managers* (henceforth DM) are (to our knowledge) non existent. Evaluation as such has been target for a lot of research. Recently more or less *automatic* testing and evaluation methods has been proposed (e.g. (Eckert et al., 1998; Scheffler and Young, 2000; Lin and Lee, 2000)).

A special problem for the development and testing of a DM is that one often has to wait until the whole system (including speech recognizer(s) and synthesis, parser/generator etc.) has been integrated. Moreover, to test the complete system one usually has to put people (e.g. the system developers or beta testers) in front of the system, feeding it with "appropriate input." Using the developers of the system as testers has the potential disadvantage that the system will just be tested with the type of phenomena or dialogues the developer has in mind. (S)he also has knowledge about the internals of the system and this can influence the testing in unpredictable ways (Araki and Doshita, 1997). Another important factor for the testing of DMs concerned with spoken input is speech recognition errors and their effects on the input.

As we started this project, the following goals and experiences guided us:

- It is **cumbersome** to test the DM with the complete system at hand. Although this testing is necessary, we would like to minimize the test effort necessary.
- We must reach a status of the DM where it is as **error free** as possible. There must not be any technical bugs in the program itself as well as logical bugs, or put in other words: The DM must not fail on any input.
- **People behave weird** (Eckert et al., 1995). To us there is no hard borderline between legal moves and non legal

* The authors wishes to thank Ralf Engel for help with the implementation and Norbert Reithinger, Tilman Becker, Christer Samuelsson and Thorsten Brantz for comments on earlier drafts and fruitful discussions.

moves in a dialogue. Some moves make more sense than others, but can the user be obliged to say only certain things at a certain point in a conversation? We think not! A dialogue system should be able to react on any input, how weird it might be.

- **Speech Recognizers makes errors.** For our dialogue system with a large vocabulary, the recognition rate drops to between 70 and 80% for certain problematic speakers. Consequently every fourth or fifth word can be wrong. An average user contribution contains 5 words in the application we refer to here (Ehrlich et al., 1997), not including single-word utterances in the calculation.

  Thus, every utterance may contain a falsely recognized word that may or may not be important for parsing or semantic construction.

To overcome some of the problems stated above and to find errors as early as possible during the course of developing a dialogue system, we have developed a validation tool - VALDIA - for the automatic testing of the DM. The overall goal we had in mind was to be able to obtain a status of the DM such that it at least does not contain any loops or other fatal (trivial) dialogue strategy errors. To become independent of the completion status of the overall system, we decided to peel the interfacing components (parser, generator,...) away from the DM. We now view the DM as a black box. This black box is then fed with *random* generated input in some interface language and we observe how the DM reacts on the given input. An important prerequisite is of course that the interface between the analysis component and the DM is defined.

At this point we would like to emphasize that our dialogue system is not modeled with "finite state dialogue structure" and "allowable syntax" for each state as described in (Scheffler and Young, 2000). In our view such a system is simple to test, since the system will just recognize those utterances it is designed to process. In such a scenario one can use the dialogue model for, e.g., enumerating every possible dialogue or generate "coherent" dialogues. On the other hand, our system puts no limits on what is allowed to say at a certain point in the dialogue, which makes the task of automatic testing non-trivial.

Ideally one would want to perform an exhaustive testing the DM with, say, all possible dialogues, i.e., sequences of user contributions and the respective system reactions. User contributions are supposed to have a maximum length in terms of semantic items. An investigation of the complexity of the number of possible utterances (in terms of combinations of semantic expressions) and resulting possible dialogues showed that for our DM, the testing task is so complex that the universe of possible semantic expressions cannot be tested in a reasonable amount of time (see Section ??).

Looking at the complexity of the task one is tempted to ask – "is it possible to exhaustively produce all possible dialogues of a certain length?" Or maybe more interesting: "can we feed the DM with all the generated dialogues?" In (Levin and Pieraccini, 1997) a sketch of a method to find good dialogue strategies was put forward. The authors argue that a dialogue system can be modeled in terms of a *state space*, an *action set* and a *strategy*. They show how one could automatically find an optimal strategy by feeding the system with all possible dialogues, or in our terminology sequences of user contributions. We took the natural continuation of this: to automatically generate user contributions or dialogues and feed them to the system, and then let the system find the optimal strategy itself. In this paper we explore some aspects and limitations of such an approach by analyzing the complexity of dialogues. We will, for instance, show that even if a dialogue manager can process one or ten or even one hundred user contribution(s) per second we cannot find an optimal strategy based on exhaustive search – the search space is simply too large!

The paper starts with a brief description of the architecture of the DM and the test envi-

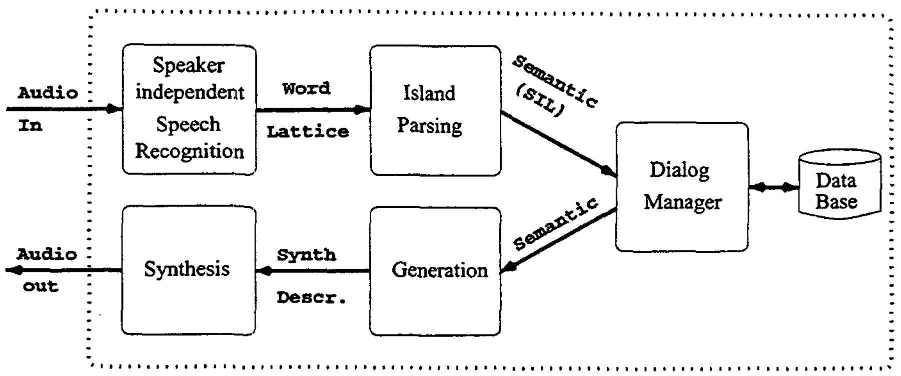


Figure 1: Schematic architecture for our dialogue system.

ronment for VALDIA, and a description of its input format. We then discuss the complexity of an utterance, continuing with the complexity of dialogues. Finally, VALDIA is described in more detail and then the paper is closed by a discussion of relevant results and papers.

2 Architecture

The dialogue system to which we first applied VALDIA (Heisterkamp and McGlashan, 1996; Ehrlich et al., 1997) was designed for answering questions about and/or selling insurances in the domain of car insurances. In case of failure or problems with the dialogue, the system passes the customer to a human operator. The architecture of the system includes an HMM-based speaker independent speech recognizer, an island parser, DM, generator and synthesizer as depicted in figure 1. The system also includes a data base which is accessed for the retrieval of domain specific information. It is important for this paper that the speech recognizer is not limited to "allowed user contributions" but outputs a word hypotheses lattice or the best chain which is processed by an island parser. Thus, the input to the DM might, depending on recognition quality, consist of arbitrary sequences of semantic expressions. A basic requirement is that the DM is not allowed to fail on any of these inputs.

For testing, we peel the interfacing components away from the DM and regard the DM as a black box. It is assumed that we send a piece of input to the DM which then reacts in a way we can observe (for instance by returning/generating some output). We assume that the DM has no notion of time. This means that to test the DM, we simply have to feed it with input and wait for it to acknowledge this by sending a responsive output request. In looking at the response, however, we have to be sensitive to effects like timeout (e.g., the DM is "thinking" too long) and/or loops (e.g., the DM outputs the same item all the time). Although in (Levin and Pieraccini, 1997) the utterances triggering the actions are not mentioned at all, this is very important. In general we don't know which utterance will trigger a certain action when the DM is in a certain state, or if the DM needs an utterance at all to perform another action. As the exhaustive validation criteria for the DM do not allow us to assume any insight into the DM itself, we have to simply feed it with all possible sequences of utterances.

Our test architecture is shown in figure 2. We connect to the DM at the same place as the analysis. We also watch the output sent to the generator. Additionally we watch the process status of the DM, that is we notice if the DM fails or breaks. In that case we can restart the DM and continue the testing.

3 Complexity

This section puts forward some notes on the complexity of dialogue. We are aware that the discussion and the results are not necessar-

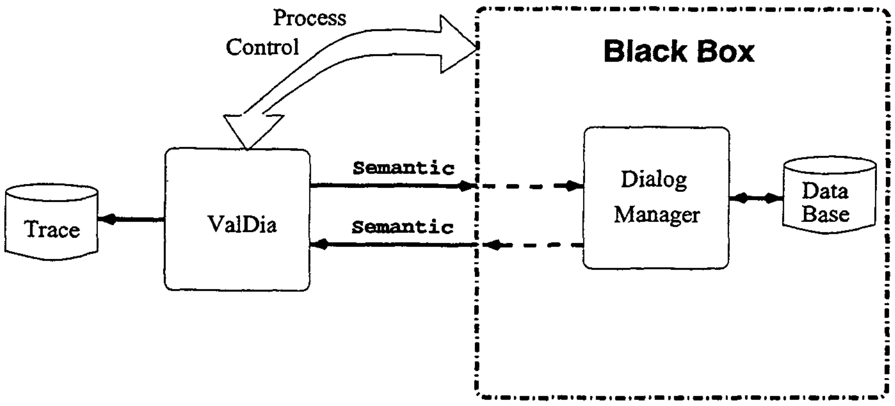


Figure 2: Schematic architecture for ValDia.

ily generalizable because they depend on the representation of the input formalism to the DM. However, we were certainly surprised by the results ourselves and it has consequences for the degree of coverage and testing one can achieve. For our dialogue system the semantic representation formalism is simple. It consists of propositional content represented as *sequences* of semantic objects the SIL[1] representation language (McGlashan et al., 1994). Here is one example: "Ein Audi 80 Avant Quattro mit ber 100 PS" *"An Audi 80 Station Wagon 4x4 with over 100 hp"*

```
 [[type:car_type,
[themake:manufacturer,
 value:audi],
   [thetype:type_name,
 value:achtzig],
   [theversion:version_name,
 value:avant],
   [thespecialfeature:feature_name,
 value:quattro],
def:indef],
  [type:power,
   themeasuretype:ps,
   thevalue:[type:number,
             cvalue:125,
     modus:[rel:above]],
modus:[rel:with]]]
```

This representation is motivated by the fact that the analysis component is an island parser (Hanrieder, 1996), and can thus find islands or sequences of semantic objects.

3.1 The complexity of an utterance

The basic entity is a *semantic object* (S) which is an atomic item treated by the DM. The DM knows about (and thus can treat or react on) M different semantic objects. Examples of a semantic object are `car_type`, `power`, `greeting`, `bye`, `integer`, and `year`. We will not pay attention to the fact that a semantic item could be instantiated with, e.g., a street name – in the navigation domain there exist about 42,000 different *names* of cities in Germany, and Berlin has 11,500 different street names – but we could of course extend the discussion below (on the cost of complexity).

We call a user contribution an *utterance*. We assume that an utterance U is a (possibly empty) *sequence* of semantic objects. This can of course be relaxed to sequences or trees in some algebra, but for this discussion it suffices to deal with sequences – as we will see, the complexity is "complex enough" with this assumption. A sentence can consist of max O number of semantic objects. An utterance is a multi-set in the real system, but for this discussion we assume an utterance is not. Each semantic object can therefore appear at most one time. Given the definitions above we can now compute the number of possible utterances $\mid U \mid$: All sequences of a certain length

[1] Semantic Interface Language

l are

$$\begin{pmatrix} M \\ l \end{pmatrix} l!$$

We therefore have

$$| U |$$
$$= \begin{pmatrix} M \\ 1 \end{pmatrix} 1! + \begin{pmatrix} M \\ 2 \end{pmatrix} 2! + \ldots + \begin{pmatrix} M \\ O \end{pmatrix} O!$$
$$= \sum_{l=1}^{O} \begin{pmatrix} M \\ l \end{pmatrix} l!$$

For one of our dialogue models, concerned with car insurance, we have $M = 25$ and $O = 9$. That is, 25 different semantic objects and we allow for a maximum of 9 semantic items (arbitrarily chosen by estimate of breath length) in one utterance:

$$| U | = 1.9 \cdot 10^9$$

Now, if we would like to test whether our DM can treat all utterances or not, we will have to wait quite a while: Suppose our DM can process 10 utterances per second, then we can process $10 \cdot 60 \cdot 60 = 36000$ utterances per hour, $36000 \cdot 24 = 864000$ utterances per day, $7 \cdot 864000 = 6048000$ per week, or $864000 \cdot 365 = 315360000$ utterances per year. To process all possible utterances we would need more than six years!

Obviously, the current parameters of the system make the complexity of the number of utterances intractable in realistic settings. Figure 3 shows how different parameter setting affects the cardinality of utterances for different values of M. The (logarithmic) y-axis represents the cardinality of utterances, and the (linear) x-axis the maximal number of semantic items in one utterance. As can be seen, for our DM, we will have to limit, e.g., the number of semantic items to 6 per utterance if we want to test all utterances in one week.

3.2 The complexity of dialogue

A dialogue can – at least theoretically – consist of a sequence of the same utterance. Many of the dialogues will of course be non-cooperative and very unnatural or, put in other words, not legal. But, as indicated above, it is important to us that the DM does not fail on *any* input. To generate all possible dialogues $| D |$ of a certain length L, we therefore have:

$$| D | = \underbrace{| U | \cdot | U | \cdot \ldots \cdot | U |}_{L \text{ times}} =$$
$$| U |^L$$

For our scenario 15 user contributions are not unnatural, so for $L = 15$ and the figures above, we have $| D | \approx 10^{140}$ which will take quite a while to process[2]. Even if we restrict the length of the dialogues to 2, we get $1.9 \cdot 10^9 \cdot 1.9 \cdot 10^9 = 3.6 \cdot 10^{18}$ theoretically possible dialogues and can thus process just an infinitely small part of them.

3.3 Consequences

Now, suppose we randomly select some dialogues out of the set of possible ones. While testing the DIALOGUE MANAGER with them we thereby encounter a certain number of (or even zero) errors, it is interesting to be able to say something about how error-free the DM is. For this discussion, it is important that by viewing the DM as a black box, we can not do anything more than assuming the errors to be distributed according to the normal distribution. Moreover, we can only apply this reasoning if we do a large number of observations. The figures below may – depending on the theoretical number of dialogues – not be valid. By using the approximation of the normal distribution we know that if we tested $N = 10000$ dialogues and received errors in DM in, say, 250 of the dialogues ($\rightsquigarrow f = \frac{250}{10000} = 0.025$), we can say that the DM contains (with a degree of confidence of 95%)

$$E = f \pm 1.96 \times \sqrt{\tfrac{f \times (1-f)}{N}} =$$
$$0.025 \pm 1.96 \times \sqrt{\tfrac{0.025 \times (1-0.025)}{10000}} \approx$$
$$0.025 \pm 0.003$$

percent errors.

In case no errors were found we get

$$E = 0 \pm 1.96 \times \sqrt{\tfrac{0 \times (1-0)}{10000}} = 0 \pm 0.$$

[2] The exact number is 2184671458940261530062771 4905000442265334978924872958985355233347509774 1304997726070386514948280700225687715652634437 757101848767098873914 3 :-)

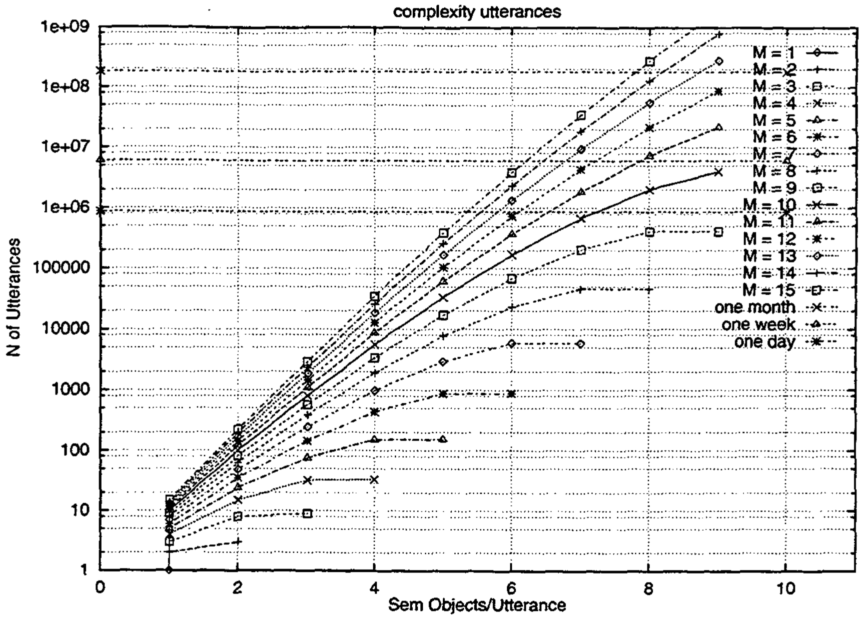


Figure 3: Utterance Complexity.

Here we have to use a trick: Instead we suppose we found one error, and thus

$$f = 1/10000 = 0.0001$$

yielding

$$E = 0.0001 \pm 1.96 \times \sqrt{\frac{0.0001\times(1-0.0001)}{10000}} = 1.96\cdot 10^{-4} \pm 1.0\cdot 10^{-6}$$

we can at least say that we are 95% confident that the DM will in *less* than

$$1.96\cdot 10^{-4} + 1.0\cdot 10^{-6} = 1.97\cdot 10^{-4}\%$$

cases raise an error.

4 ValDia – The Implementation

To allow for intelligent testing, we decided to implement our test tool in using the following three parts:

- the core test engine,
- the interface to the DM (implemented in OZ/MOZART[3]), and
- a graphical editor for the definition of stochastic automata (implemented in Tcl/Tk),

The core test engine uses the definition of stochastic automata to create sequences of semantic expressions to be sent to the DM. It records both the input and the output to and from the DM and checks for special messages (e.g. end of dialogue), crashes, if the DM is emitting the same response all the time, or other events events that indicate erroneous behaviour of the DM. It also creates test profiles and checkpoint files to enable interruption and restart of test runs.

The interface handles the connection between ValDia and the DM. It realizes a TCP/IP connection to and from the DM. In case parallel test runs are made, it can also handle different processes.

The motivation for the stochastic automaton editor and, at the same time, the main feature of ValDia (see Figure 4) is that it allows for the design of utterances or even dialogues or utterance sequences, and thus test specific areas in the space of theoretically possible dialogues. The dialogue system devel-

[3] The reason for using OZ is manifold: OZ features threads, multiple platforms (UNIX/LINUX and Windows), unification, a Tcl/Tk library, and finally it comes for free. See http://www.mozart-oz.org

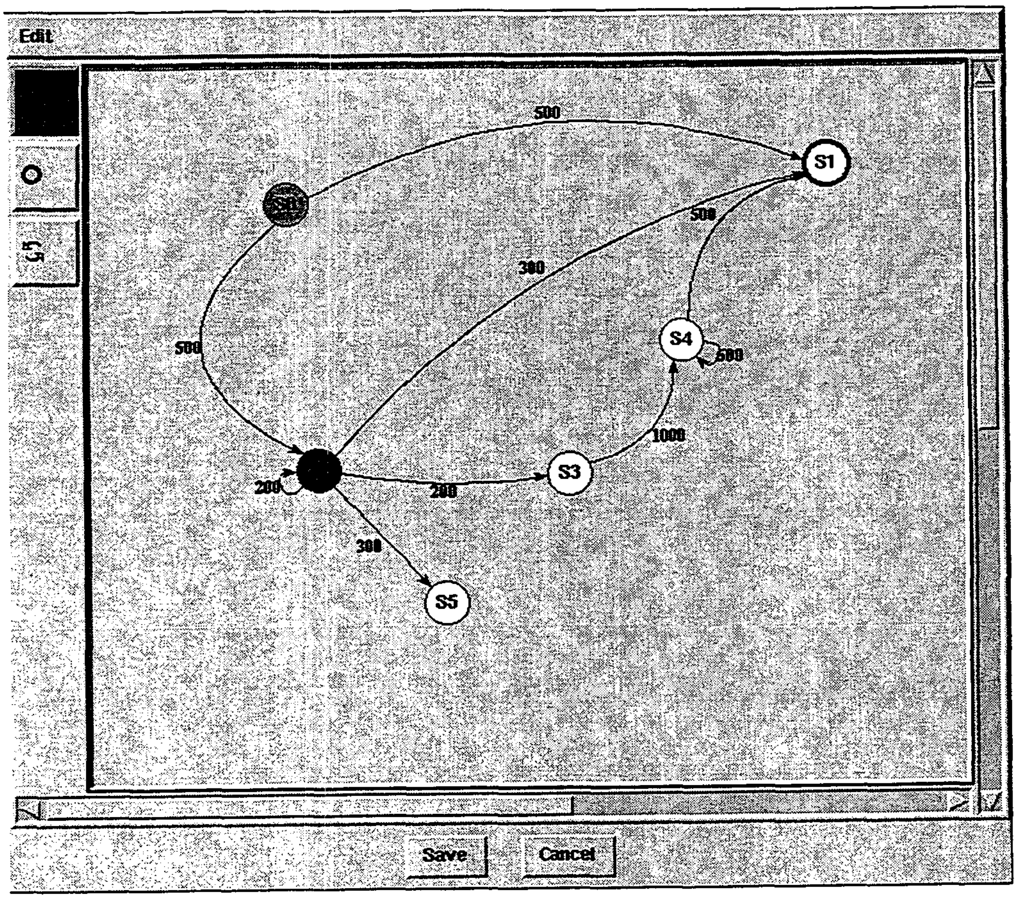
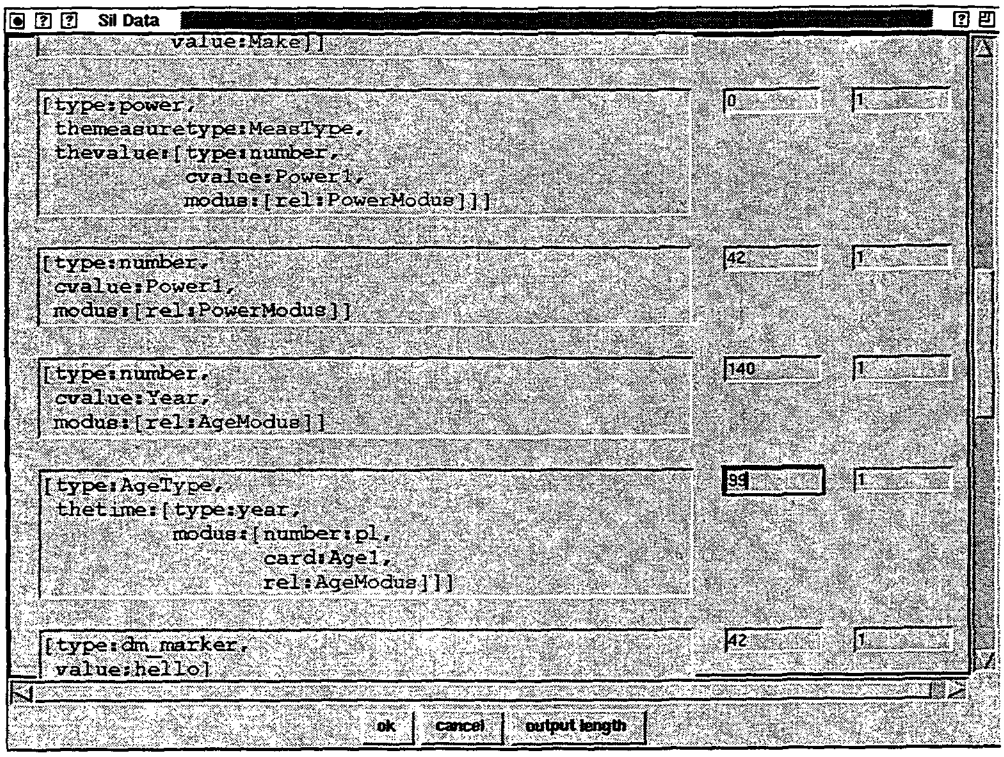


Figure 4: Screen shot of the automata editor

oper can interactively define the automata, using the pointing device to draw the states an the transitions. In each state, it is possible to change the constraints for the definition of a SIL expression. More precisely we change the *probability* of the alternatives of (a part of) an expression. The arcs between the states are augmented with probabilities which guide state transitions in a stochastic manner, thus creating certain sequences by preference, without completely excluding others. In Figure 5 the left row contains the basic semantic entities, the middle the probability, and the right one the number of occurrences for that particular semantic item in each utterance. For the semantic items the variable parts are linked to another window where the their instantiations are described. The constraints are semi-automatically derived from the definition of the interface specification for the DM. The reason for "semi-automatically" and not automatically is that we have had no time to write a generic function for this. But, basically the derivation is straightforward. Consequently we can design interesting utterance sequences, according to, e.g., experiences gained during WOZ-experiments.



```
          value:Make]]

[type:power,                                        0      1
 themeasuretype:MeasType,
 thevalue:[type:number,
           cvalue:Power1,
           modus:[rel:PowerModus]]]

[type:number,                                       42     1
 cvalue:Power1,
 modus:[rel:PowerModus]]

[type:number,                                       140    1
 cvalue:Year,
 modus:[rel:AgeModus]]

[type:AgeType,                                      99     1
 thetime:[type:year,
          modus:[number:pl,
                 card:Age1,
                 rel:AgeModus]]]

[type:dm_marker,                                    42     1
 value:hello]
```

Figure 5: Part of the constraints of an utterance

Finally, by using just one state and no constraints, we can, of course, produce completely arbitrary utterance sequences.

During the testing of the dialogue manager we can run the system in two modes. The first - exhaustive mode - generates all sequences of dialogues by enumerating all dialogues. This is based on the enumeration of all possible utterances in each state. The exhaustive mode can be used when we know that the complexity of the automaton (and utterances) is testable - VALDIA can compute the number of dialogues and compute an upper time limit based on the computational power of the DM. In the second mode - Monte Carlo mode - the utterance generation in each state as well as the change of state is random. In this way we randomly walks the automaton and randomly generate utterance profiles. This has been proven useful in the cases where we number of possible dialogues to large is for exhaustive testing.

Notice that we can not pay any attention to legal moves. VALDIA has (i) no knowledge about what a legal move is, and (ii) no possibility to react on the response from the DM. Therefore the "legal moves" and "cooperativeness" is non existent concepts here. But, this is what we want: People behave weird! Our speech recognizer produces errors! And most important: We have to live with this, and must not fail on any input!

5 First Results

During the development of VALDIA we have detected several errors in the implementation of our DM. Most of the errors where logical errors of the kind "Now that's a combination of things we didn't cover." e.g., the co-occurence of good_bye and request_repetition in a user utterance led to a goal conflict in the DM that caused it to hang, as did the non-exclusive handling of disjunction in *"It's older (or) younger than 5 years"*, etc.

Additionally we discovered that the DM in some of the test runs crashed after about 500 (!) dialogues due to erroneous memory handling. This is something one would never detect during normal testing with a full system, but immediately *after* delivering the system.

VALDIA produces huge amounts of (huge) trace files. Analyzing these is at present a pain as big as testing the complete dialogue system. Consequently, we will have to develop functionality for condensing the trace information.

6 Conclusion

The project VALDIA has produced useful insights into the complexity of dialogue: Spoken dialogue is very complex! Exhaustive testing of a DM is for some scenarios/dialogue models impossible. The results were obtained during the development of a test program for a DM. Purpose of the testing was to be able to integrate a DM into the dialogue system which contained as few errors as possible. We would like to highlight the following points:

- VALDIA has proven its usefulness in that it is able to detect errors in the implementation of DMs *before* it is integrated into the complete dialogue system. During the testing we encountered, in addition to logical bugs, errors which would never be detected during normal testing with the complete dialogue system.

- By including the automata into VALDIA it is possible to concentrate the testing on "interesting utterance sequences" and, despite the huge universe of theoretically possible dialogues, obtain a status of the DM which for certain tasks is well tested.

- It is simple to adapt for the testing of a new DM. Technically the only thing that has to changed is the definition/constraints of the definition utterances. This is at present a semi-automatic process. Conceptually the automata has to be defined, unless one wants to test in Monte Carlo mode.

- In the current implementation VALDIA uses about 10% of the processing time compared to the DM. Thus VALDIA can control between 5 and 10 instances of the DM depending on available resources in the net.

- VALDIA is platform independent. At our site, we are using a mixture of different types of computers, both PCs running under Windows/Linux and UNIX machines. Depending on load, we are flexible to utilize any of the free resources for the testing.

We are currently in the process of adapting VALDIA for a new scenario. For this DM input consists of grammatical structures, rather than sets of semantic objects. Since the VALDIA project started, interesting research results have emerged and there are lot of things that remain to be done. Amongst those, we will pay attention to at least the following topics:

- The current implementation of VALDIA has no means of react on the output from the DM. For intelligent testing this has to be incorporated into the system. Possible future directions are described in (Eckert et al., 1998), (Scheffler and Young, 2000) and (Lin and Lee, 2000). In, e.g., (Eckert et al., 1998) VALDIA is replaced by an *simulated user*, and the authors describe a statistical method for reacting on system responses.

- We have to develop a tool for semi-automatically analyzing the trace files produced by VALDIA. Possible future features are just saving the files of those dialogues/utterances which resulted in an error.

References

Masahiro Araki and Shuji Doshita. 1997. Automatic evaluation environment for spoken dialogue systems. In Elisabeth Maier, Marion Mast, and Susann LuperFoy, editors, *Revised papers from the ECAI-96 Workshop in Budapest, Hungary on Dialogue Processing in Spoken Language Systems*, Heidelberg, August. Lecture Notes in Artificial Intelligence, Springer-Verlag.

Wieland Eckert, Elmar Nöth, Heinrich Niemann, and Ernst-Günter Schukat-Talamazzini. 1995. Real users behave weird - experiences made collecting large human-machine-dialog corpora. In Paul Dalsgaard, Lars Bo Larsen, Louis Boves, and Ib Thomsen, editors, *Proceedings of ESCA Tutorial and Research Workshop on Spoken Dialogue Systems '95*, Vigsö, Denmark.

Wieland Eckert, Esther Levin, and Roberto Pieraccini. 1998. Automatic Evaluation of Spoken Dialogue System. Technical report, AT&T. Technical Report Nr. TR98.9.1.

Ute Ehrlich, Gerhard Hanrieder, Ludwig Hitzenberger, Paul Heisterkamp, Klaus Mecklenburg, and Peter Regel-Brietzmann. 1997. Access - automated call center through speech understanding system. In *Proceedings of Eurospeech '97*, Rhodes.

Gerhard Hanrieder. 1996. *Inkrementelles Parsing gesprochener Sprache mit einer linksassoziativen Unifikationsgrammatik.* Ph.D. thesis, Universität Erlangen-Nürnberg. http://www.infix.com – ISBN 3-89838-140-4.

Paul Heisterkamp and Scott McGlashan. 1996. Units of Dialogue Management: An Example. In *Proceedings of* ICSLP*-96*, Philadelphia, PA, October.

Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *Proceedings of EuroSpeech-97*, Rhodes.

Bor-schen Lin and Lin-shan Lee. 2000. Fundamental performance analysis for spoken dialogue system based on a quantitative simulation approach. In *Proceedings of ICASSP-2000*, Istanbul, Turkey, June 5–9.

Scott McGlashan, Francois Andry, and Gerhard Niedermair. 1994. A Proposal for SIL. Technical report, University of Surrey, CAP SOGETI, and Siemens AG, March. SUNDIAL report.

Konrad Scheffler and Steve Young. 2000. Probablistic simulation of human-machine dialogues. In *Proceedings of ICASSP-2000*, Istanbul, Turkey, June 5–9.