Rapid Dialogue Prototyping Methodology

Trung H. BUI, Martin RAJMAN
Artificial Intelligence Laboratory (LIA)
School of Computer and Communication Sciences (I&C),
Swiss Federal Institute of Technology Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
{trung.bui, martin.rajman}@epfl.ch

January 5, 2004

EPFL Technical Report IC/2004/01

Abstract

The objective of this document is to present a rapid dialogue prototyping methodology developed at the Artificial Intelligent Laboratory - Ecole Polytechnique Federale de Lausanne. Concretely, the rapid dialogue prototyping methodology is decomposed into 5 consecutive main steps: (1) producing the task model; (2) deriving the initial dialogue model; (3) using a Wizard-of-Oz experiment to instantiate the initial dialogue model; (4) using an internal field test to refine the dialogue model; and (5) using an external field test to evaluate the final dialogue model.

Keywords: Dialogue Modeling and Dialogue Management, Wizard-of-Oz

Contents

1 Introduction **3**

2 Task Model **3**

3 Dialogue Model **4**
3.1 Definition 4
3.2 Generic Dialogue Nodes 5
3.2.1 Prompts 6
3.2.2 Grammars 6
3.3 Local Dialogue Flow Management Strategy 7
3.4 Global Dialogue Flow Management Strategy 7
3.4.1 Branching Logic 8
3.4.2 Dialogue Dead-end Management 10
3.4.3 Confirmation 10
3.4.4 Dialogue Termination 12
3.4.5 Incoherences 12
3.5 Generating the initial model 12

4 Wizard-of-Oz experiment **13**

5 Internal Field Test **15**

6 External Field Test **15**

7 Conclusion **16**

1 Introduction

Human-machine communication has been the goal of researchers for more than 30 years. Many approaches to dialogue systems have been implemented and many surveys on this topic have been produced (e.g. [McTear, 2002, Catizone *et al.*, 2002, Churcher *et al.*, 1997, Cohen, 1997]). Up to now, due to the complexity of the management of spoken language interfaces and their strong dependence on the interaction context, there does not exist yet a really generic approach for dialogue design; each application requires the development of a specific model. Dialogue prototyping therefore represents a significant part in the development process of interactive systems, especially for the ones with a vocal interface: there is a strong need for an efficient Rapid Dialogue Prototyping Methodology (RDPM). The main idea of the methodology is to quickly produce a deployable dialogue model and its improvement through a iterative process based on Wizard-of-Oz experiments (i.e. dialogue simulation) [Fraser and Gilbert, 1991].

In this perspective, the main goal of the document is to describe the RDPM[1] developed at the Artificial Intelligent Laboratory (LIA) - Ecole Polytechnique Federale de Lausanne (EPFL). Concretely, the RPDM contains five main steps:

1. *producing a task model* for the targeted application;
2. *deriving an initial dialogue model* from the obtained task model;
3. *carrying out a Wizard-of-Oz experiment* to improve the initial dialogue model;
4. *carrying out an internal field test* to further refine the dialogue model (reformulation of system messages, improved feedback, etc.), and to validate the evaluation procedure (coherence, understandability); and
5. *carrying out an external field test* to evaluate the final dialogue model according to the evaluation procedure defined during the internal field test.

All 5 steps will be presented more detail in the next sections.

2 Task Model

In the RDPM, a task model is described in the form of a set of relational tables (frames), where the columns are the attributes needed to identify the tasks to be performed and the lines are the possible task instances (also called the "solutions" or the "targets").

More precisely, a task is modeled as a function, the arguments of which correspond to the above-mentioned attributes and the call to which results in the fulfillment of the task, or in other words, the selection of the task instance ("solution") corresponding to the values selected for the attributes. For example, in the InfoVox [2] restaurant application, the task model simply reduces to a single function `select_restaurant(Type_of_kitchen, Localization, Opening_time, Opening_day, Price_range)`, the attributes of which identify the 5 selection features available for the restaurant search. Therefore, the task model of the Infovox project is simply a table with 5 attributes: `Type_of_kitchen`,

[1] some preliminary description of this methodology is presented in [Rajman *et al.*, 2003]

[2] The InfoVox project was partly funded by the Swiss national CTI grant program.

`Localization`, `Opening_time`, `Opening_day`, and `Price`, the lines of which are the various value combinations of the attributes corresponding to existing restaurants.

At the computational level, the calls to the `select_restaurant()` function were implemented in the form of SQL queries to be submitted to the project database containing the required information about the around 50 available restaurants in the city of Martigny (the values of the selection features of course, but also additional information such as the name, the address and the telephone number of the restaurants).

Notice that the current version of the RDPM presupposes that the task model consists of a single table called the *complete solution table*. In the case of more complex models consisting of several interconnected tables (for example a main table containing the task candidates and several additional tables relating the values present in the main table to additional attributes), standard database normalization procedures (such as joint operations) are first applied to transform the original tables in to a single one. Figure 1 gives a simple illustration of such a transformation.

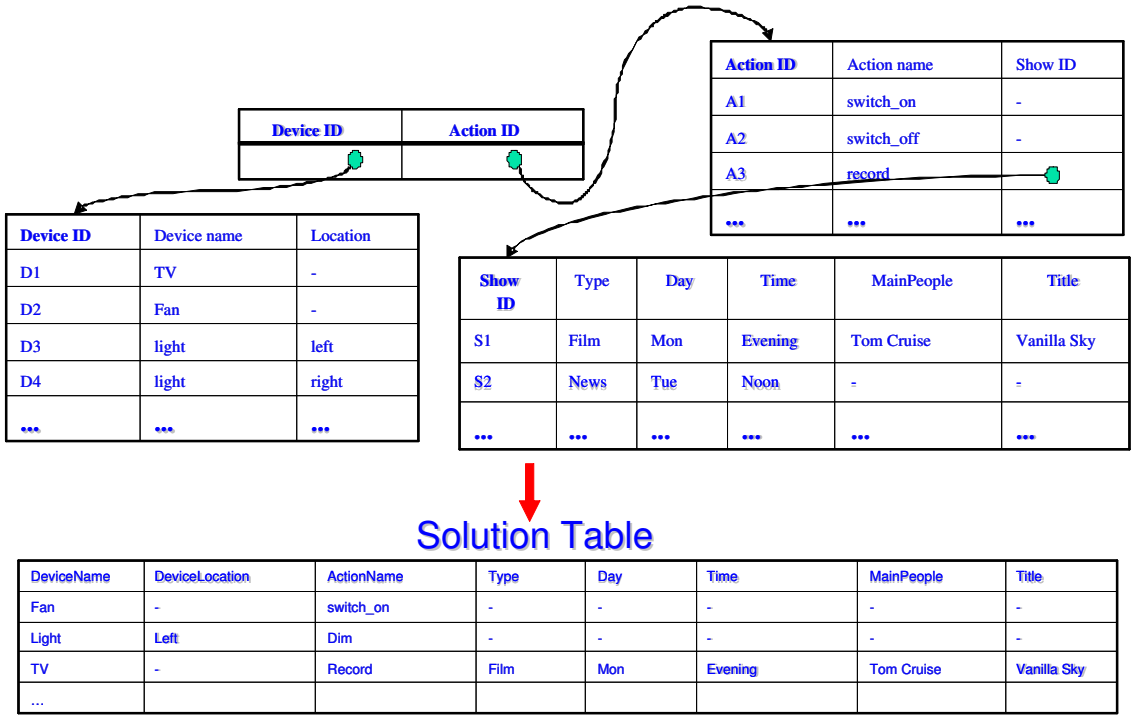

Figure 1: *From the task model to the complete solution table*

3 Dialogue Model

3.1 Definition

In our approach, a *dialogue model* is defined as a set of interconnected Generic Dialogue Nodes (often hereafter referred to as GDNs, e.g. [Bilange, 1992]), where each of the dialogue nodes is associated with one of the attributes (also called "slots" or "fields" hereafter) in the solution table. For any given slot, the role of the associated generic dialogue node is to perform the simple interaction with the user that is required to obtain a valid value for the associated attribute.

In the architecture that we have selected for the implementation of our dialogue models, the processing of the GDNs (i.e. the actual interaction with the user according to the specification of the GDNs) is taken in charge by a specific module called the *local dialogue manager*. However, this is of course not sufficient to perform any real dialogue, as some form of global dialogue management also needs to be integrated. For example, in addition to the definition of the GDNs and the specification of the local dialogue manager, some branching logic responsible for the management of the global dialogue flow needs to be specified. In our approach, this branching logic is hard-coded in a specific dialogue management module called the *global dialogue manager*. The underlying assumption is that the encoded local and global management strategies are indeed application independent, i.e. that, in most cases, they lead to an acceptable, even not always optimal behavior for the system. Consequently, in our approach, dialogue model design essentially reduces to the application dependent, declarative specification of the GDNs, the encoded dialogue management strategies being used without modification for all applications.

In short, a dialogue model consists of 2 main parts: (1) the application dependent declarative specification of the GDNs; and (2) the application independent local and global dialogue flow management strategies encoded in the corresponding dialogue managers. Both of these 2 components are described in more detail in the next sections.

3.2 Generic Dialogue Nodes

To deal with the various attributes appearing in the relational tables defining the task model, we consider two main types of GDNs:

1. Simple GDNs (also called Static GDNs) associated with *Static fields* (i.e. fields the values of which do not change in time, or change only very slowly; for example the names of the devices that can be operated in a given room);

2. List processing GDNs (also called Dynamic GDNs) associated with *Dynamic fields* (i.e. fields the values of which quickly change in time; for example the list of films that might be recorded).

In addition, a third type of GDNs called "Internal GDNs" is used to perform the interactions that are required by various special functions implemented in the dialogue manager (e.g. start/reset the dialogue).

As already mentioned, the role of each GDN is to perform a simple interaction with the user to obtain a valid value for the associated attribute. In this respect, the difference between static and dynamic GDNs is that the former are expecting the user to directly provide a value for the associated attribute (for example, a static GDN associated with the "Device Name" attribute might ask a question such as "What device would you like to operate?" and will be expecting an answer containing a value taken from a predefined list of values such as "fan", "tv" or "answering machine"), while a dynamic GDN will ask the user to choose in a list of dynamically computed list of values (for example, a dynamic GDN associated with the "Film Title" attribute will generate a list of titles and ask the user to indicate the position of the selected one in the list). The List processing GDNs are an important component of the targeted dialogue model as they allow to efficiently take into account large dynamic vocabularies that could not be reliably processed by Simple GDNs because of the limited performance of the speech recognition module in such conditions.

To realize the interaction it is responsible for, each GDN contains 2 main types of components: prompts and grammars (Figure 2).

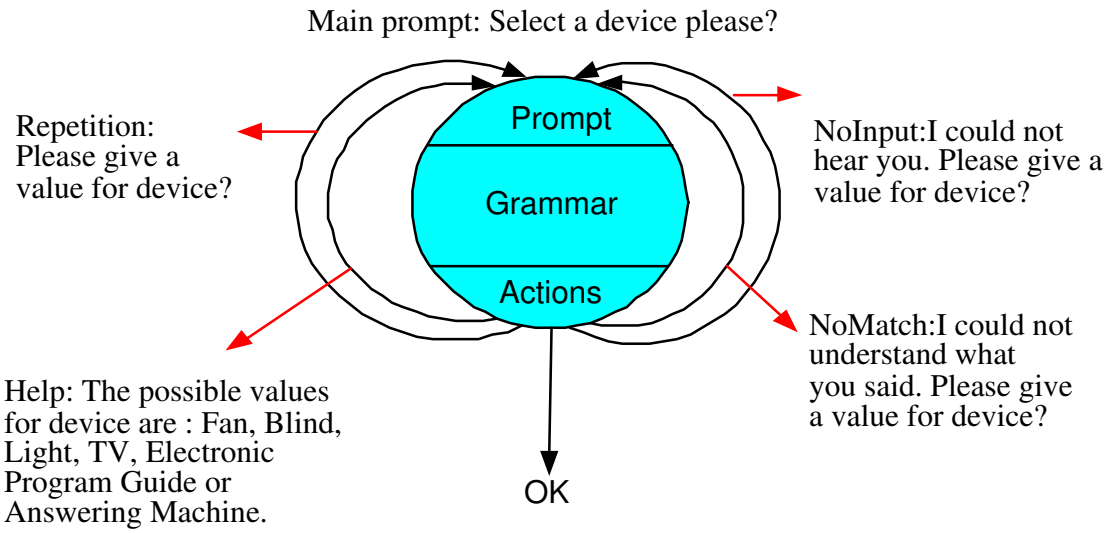

Figure 2: *An example of the GDN "Device Name"*

3.2.1 Prompts

Prompts are the messages uttered by the GDN during the interaction. Several types of prompts are defined, among which them the main prompt corresponding to the initial question asked by the GDN and the help prompt that is uttered in the case of a request for help expressed by the user. Notice that the formulation of the prompts plays an important role during the dialogue as it influences the level of *mixed initiative* (i.e. the degree of flexibility that the system allows for the interaction). For instance, a main prompt such as "What can I do for you?" expresses the fact that the system is ready to accept a quite a broad range of user requests, while a more precise prompt such as "Do you want to operate the TV, yes or no?" implies a low level of mixed initiative as the user is only expected to answer either yes or no.

3.2.2 Grammars

The role of the grammars is to make the connection between the surface forms appearing in the natural language user utterances and the "canonical values" used in the task model, that is the set of values defined for the attributes associated with the GDNs in the solution table describing the application. As such, the grammars represent the main Natural Language Processing elements in the system. The grammars might also be used in the speech recognition engine to improve the quality of the recognition. In addition, the control of the level of mixed initiative is implemented through the notion of *active grammars*: in its specification, each GDN is associated with a set of grammars that define the types of answers that are considered as acceptable for the interaction the GDN is responsible for. For example, the GDN associated with "Show Time" field will of course be associated

with the "Show Time" grammar (recognizing utterances such as "at 8 o'clock", "in the afternoon") but might also be associated with other grammars such as the "Show Date" grammar in order to be able to extract, for an utterance such as "tomorrow evening" not only the time ("evening") but also the date ("tomorrow").

3.3 Local Dialogue Flow Management Strategy

Each GDN is able to locally process five types of possible generic situations: (1) *OK*: the user answers the question in an acceptable way; (2) *Request for Repetition*: the user asks for the repetition of the last system prompt; (3) *Request for Help*: the user doesn't know how to answer to the question and asks for more explanation; (4) *NoInput*: the user provides no utterance; and (5) *NoMatch*: the user answers but nothing useful can be extracted from his/her answer.

In the case of the *OK* situation, the control is handed back to the global dialogue manager which applies the global dialogue management strategy for the activation of the next GDN. In the other 4 situations, the control remains at the GDN level. In these "problematic" cases, there is therefore a need for repairing the dialogue and the system then operates in the following way: (a) *Request for Repetition*: the current GDN is reactivated and its main prompt is played if it is the first request, otherwise the reformulation prompt is played; (b) *Request for Help*: the GDN is reactivated and the associated help prompt is played instead of the main prompt; and (c) *NoInput/NoMatch*: the current GDN is reactivated and the NoInput/NoMatch prompt is concatenated at the beginning of the main prompt.

Notice that, in all cases, there is an upper limit to the number of consecutive times that a given GDN can be activated. If this limit is exceeded, the control is handed back to the global dialogue manager with the appropriate error message.

3.4 Global Dialogue Flow Management Strategy

The Global Dialogue Flow Management (GDFM) consists of several complementary strategies:

- a branching logic defining the next GDN to be activated;
- a dialogue dead end management strategy to deal with dialogue situations where no solution corresponds to the request expressed by the user;
- a confirmation strategy to provide the systems with validation possibilities for the values acquired during the interaction;
- a dialogue termination strategy to define when the interaction with the user should be terminated (i.e. a solution proposed); and
- a strategy to deal with incoherencies.

As already mentioned, all these strategies are encoded in the global dialogue manager and are therefore application independent.

3.4.1 Branching Logic

The proposed branching logic only relies on the fact that the task model is expressed in the form of a relational table. It can be described as the following 4 steps process:

1. Acquire: some canonical values are obtained from the user through the interaction with the current GDN level;
2. Filter: the obtained values are added to the set of already acquired ones and the application database is filtered in order to contain only the solutions that are compatible with the obtained set of values;
3. Propagate: for the attributes for which all the solutions in the database have the same canonical value, the value is propagated, i.e. considered as "implicitly" acquired for the attribute;
4. Activate: the next "open" attribute (i.e. the next attribute still associated with a heterogenous of values) is identified, and the associated GDN is activated.

Example:
System[1]: What do you want to do?
User[1]: record a show on TV.
System[2]: For TV as device and record as action, which day of week?

1. Acquire: The acquired information from the *User[1]* utterance are: DeviceName = "TV", ActionName="Record";
2. Filter: Let us assume that the complete solution table contains only the following 3 rows compatible with the above acquired values:

DeviceName	DeviceLocation	ActionName	Type	Day
TV	-	Record	Film	Sun
TV	-	Record	Film	Sat
TV	-	Record	Film	Mon

   After filtering, the current solution table will therefore reduce to these 3 solutions only.
3. Propagate: the value "-" and "film" are propagated for the attributes DeviceLocation and Type respectively;
4. Activate: the next "open" attribute is "Day" (still active with 3 different values) and the associated GDN therefore become the new active one and the question *System[2]* is asked.

All 4 above steps are illustrated in the figures 3, 4.

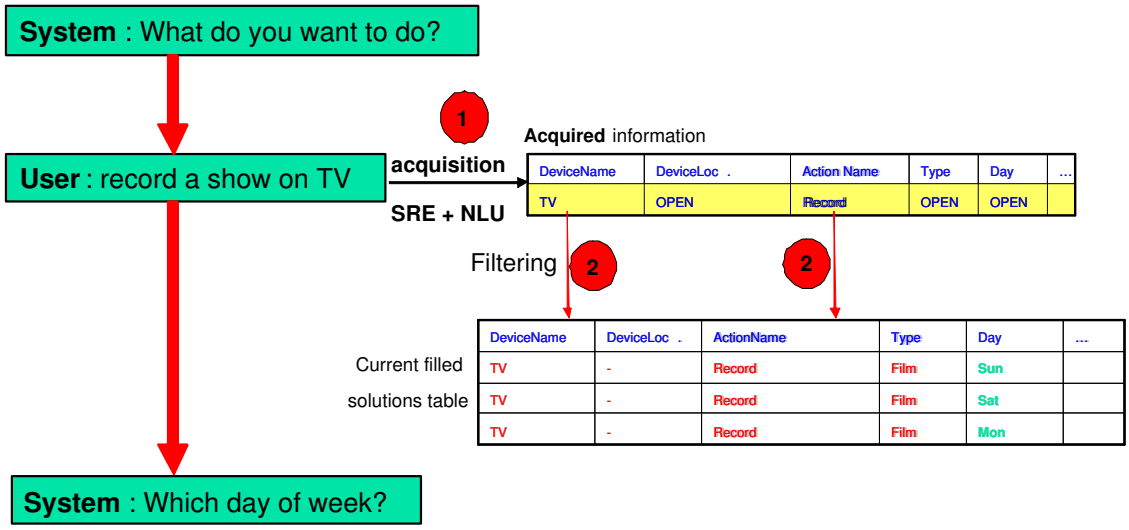

Figure 3: *Branching Logic (Steps 1, 2)*

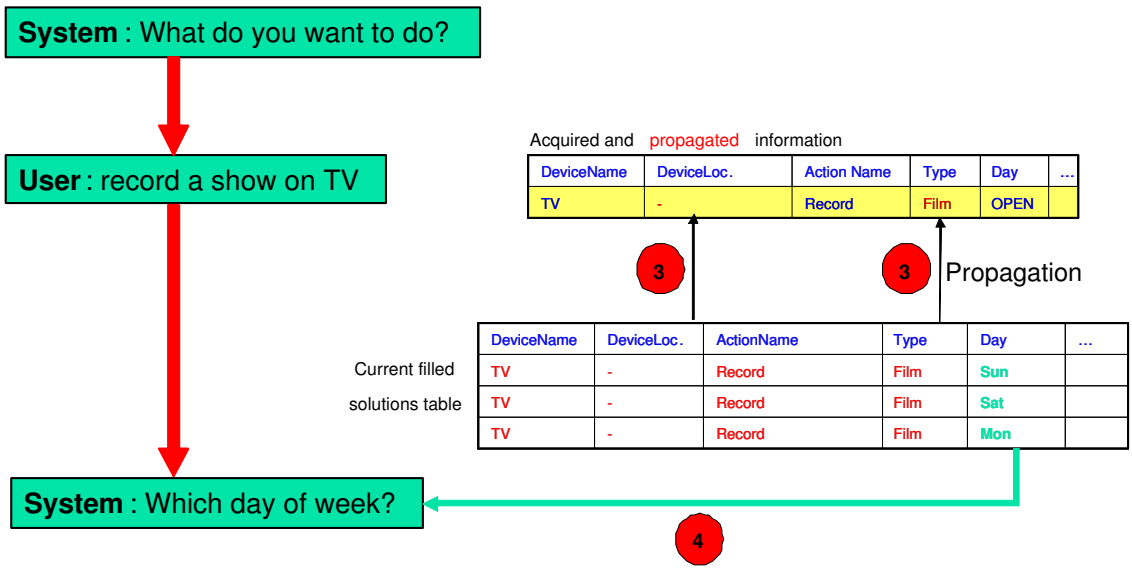

Figure 4: *Branching Logic (Steps 3, 4)*

3.4.2 Dialogue Dead-end Management

This strategy is required to deal with the cases where the goal of the dialogue can not be reached any more (zero solution). To cope with dead end situations, we use the following relaxation strategy:

1. Determine how many solutions are compatible with all the values that have been explicitly acquired (i.e. not propagated) but one. If the obtained number is smaller than or equal to a predefined threshold called the "dead end management threshold" then provide all the relaxed solutions to the user and ask him/her to select the desired one. Otherwise choose one of the attributes corresponding to a non-zero number of solutions when relaxed;
2. Remove the value associated with the selected attribute, re-propagate from the remaining ones, and activate a yes/no GDN to get user's decision about the relaxation;
3. If the user agrees with the relaxation, activate the next GDN according to the standard activation rule, otherwise go to step 2;
4. If the user rejects all relaxation possibilities, reset the dialogue.

Example:
System[1]: What do you want to do?
User[1]: dim the fan

As this stage of the dialogue, the system has acquired the value "dim" for "ActionName" and "fan" for "DeviceName" and, as it is assumed that it is not possible to dim a fan, the system is in a dead end situation. It then first relaxes "dim" and computes the number of solutions compatible with the unique constraint "DeviceName=fan" (say 2 entries) and then relaxes "fan" and computes the number of solutions compatible with "ActionName=dim" (say 3 entries). The total number of relaxed solutions is therefore 5 and:

- if the dead end management threshold is greater or equal to 5, the system will display/utter the solutions and ask the user to select one (see Figure 5):

  System[2]: Your selection does not correspond to a possible solution. Please select a possible solution from the list?

- otherwise, the system will start the relaxation confirmation process:

  System[2]: Your selection does not correspond to a possible solution. Do you agree to reconsider "fan" as device?

3.4.3 Confirmation

The confirmation strategy is the procedure used during the dialogue to obtain the user's confirmation of the values that have been acquired by the system. There are 2 possible approaches:

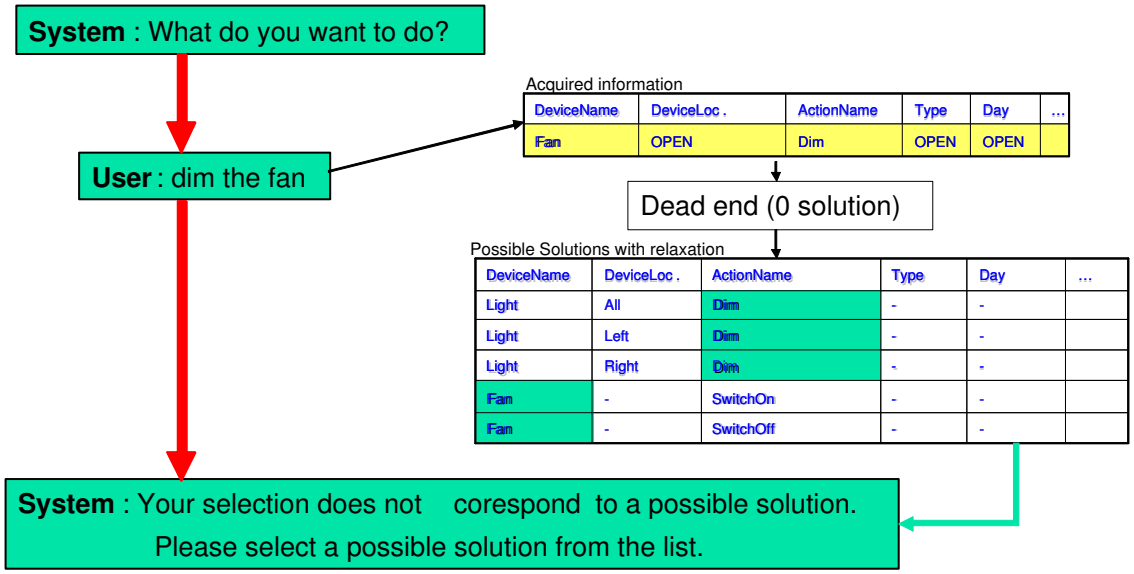

Figure 5: *Example of Dead end Management*

- Explicit confirmation: the confirmation is simply obtained by explicitly asking the user;

  Example:
  System[1]: Now you can select your show from the list.
  User[1]: I select the first one
  System[2]: You have selected solution one. Is it correct?

- Implicit confirmation: the confirmation is induced from the reaction of the user to some confirmation information automatically associated with the next question.

  Example:
  System[1]: Select a device please?
  User[1]: Fan
  System[2]: For Fan as device, what do you want me to do?
  User[1]: Switch it on.

  In this example, the fact that the user did not react negatively to the indication "For fan as device,..." is interpreted as an (implicit) positive confirmation. The underlying hypothesis is that the user would have reacted in a negative way (by saying something like: "I didn't say I want to operate the fan!...") if the proposed choice would not have been correct.

Implicit confirmation usually leads to shorter dialogue that are considered as more natural by the users. Explicit confirmation is useful in special cases such as the invocation of irreversible actions (such as "delete the messages on the answering machine".

3.4.4 Dialogue Termination

The idea behind the dialogue termination strategy is that it might be more efficient, once a limited number of solutions has been reached during a dialogue, to simply display/utter the solution list and let the user choose the correct one, instead of trying to continue the dialogue to refine the user request in order to reduce the solution set to a unique one. For example, when the user wants to record a film on TV at a specific time, and the number of available films at this time is sufficiently small, the system will display all the possibilities and ask the user to select a film instead of asking for additional selection criteria (e.g. the main actors). To implement such a behavior, a limit on the number of current solutions, called the *termination threshold* is defined and the termination strategy then very simply corresponds to stop the dialogue and switch to explicit solution selection as soon as the current number of solutions is smaller than or equal to the termination threshold. Once the selection is done, the task is completed and the dialogue is reset.

3.4.5 Incoherences

This strategy is necessary to deal with the cases where the user provides two incompatible values for one or several attribute(s). For example, if the user first indicates *"fan"* for the Device Name attribute and later on provides the value *"light"*, an incoherence occurs for this attribute. To deal with such a situation, an incoherence prompt, such as, for example: *"I am not sure about the device you have provided: Fan or TV. Could you please repeat your choice?"* should be generated in order to force the user to make an explicit choice.

The above mentioned incoherence management strategy is only used for incompatible value pairs where each of the two values has been explicitly provided by the user ("true" incoherences). If only propagated values are involved, the new value is used to overwrite the old one. In the remaining cases (propagated against given or vice versa), a dialogue dead-end management is triggered.

In the case of several simultaneous incoherencies, only one is processed and all other incoherent values (old and new) are removed. The rule to choose the incoherence pair to process is the following:

1. If the current GDN defines a context (i.e. is an associated with a specific attribute on which the current question was focused) and if there is an incoherence associated with that attribute, then this incoherence should be processed;

2. Otherwise (the case of GDNs corresponding to fully open questions such as: "What you want to do?"), the incoherence corresponding to the attribute associated with the GDN coming first in the order defined in the solution table should be processed.

3.5 Generating the initial model

Each attribute in the complete solution table is associated with a GDN in the dialogue model, the type (static or dynamic) of which is derived from the type of the attribute.

Notice that, the dialogue model might also contain some GDNs that are not explicitly associated with an attribute in the complete solution table. For example, the initial (mixed initiative) start GDN as we mentioned in 3.2.

The generated GDNs are minimalistic in the sense that their prompts are automatically generated and that they do not contain any interpretation grammars. The generated prompts are produced in a very simple way: if the attribute to be queried about is X, then the associated question is simply: "What X?". Following this trivial generation procedure, we can now produce, as the final part of our illustration, an example of what could be an interaction between a user and the automatically generated dialogue model described in this section:

System1: What device?
User[1]: I would like to use the VCR.
System[2]: What action?
User[2]: Switch it on to record "The Simpson's".
System[3]: Which day?
User[3]: Tomorrow.
System[4]: What Start time?
User[4]: From 7 to 7:30pm
System[5]: Switching on the VCR to record "The Simpson's" on Friday, January 31st, from 7 to 7:30pm.

In reality of course, as the generated dialogue model does not contain any interpretation grammars, it will in fact not be able to produce an interaction such as the one shown above, because it will simply be unable to interpret any of the user's answers. In this stage, the validation of the model therefore still requires human intervention. This is done in the framework of a Wizard-of-Oz experiment as described in the next section.

4 Wizard-of-Oz experiment

A Wizard-of-Oz experiment [Fraser and Gilbert, 1991] (hereafter called a WoZ experiment) can be defined as a simulation of a human-machine interaction during which a user is exposed to a system he/she believes to be fully automatic, while a hidden human operator (the Wizard) is manually operating (at least) some of the system functionalities that have not yet been fully implemented (sometimes, no implementation at all has been done at the WoZ stage and the experiment then corresponds to a complete simulation) [Dahlbäck and Ahrenberg, 1993, Geutner Petra and Dietrich, 2002].

Within such a setup, the main goal of a WoZ experiment is to provide "realistic" experimental data about the "true" behaviour of the members of some targeted user group when faced with an automated system for a given application. To this end, the experimental data is gathered (the experiments are often recorded and/or filmed) and subjectively analysed by the system designers in order to obtain valuable insights to guide subsequent modelling and implementation decisions [Daly-Jones *et al.*, 1999]. The underlying hypothesis is that it is easier (quicker, cheaper, ...) to set-up and modify a manually operated simulation than to specify, develop, and modify a real implementation of a system. While this hypothesis is undoubtfully very often true, it is however important to notice that a WoZ experiment is certainly not a cheap operation. It is indeed not an easy task to make users convincingly believe that they are faced with a machine when the simulation is in fact operated by a human. All clues that could reveal the presence or intervention of the wizard must be thoroughly removed. Thus, actions need to be

taken to physically hide the wizard during the interaction and an interaction interface, even simplified, usually needs to be developed to this end. Further, the wizard has to undergo a specific training so that he/she can consistently behave in a manner that can convincingly be believed as the one the user is expecting from a machine (no sophisticated inferences, no emotional reactions, no apparent tiredness, ...). In addition, it can be quite difficult to guarantee that the behaviour of the simulated system will remain uniform over time (the wizard can be in better shape one day than the other, he/she might not consistently remember the provided instructions to operate the simulation over a longer period of time, ...) [McTear, 2002].

However, it is also clear that a WoZ experiment can significantly improve the design of an interactive system (e.g. the design of the user interface [Boyce and Gorin, 1996]). Indeed, the results of the experiments can not only be used for an initial evaluation of the adequacy of the a priori conception (architecture, functions, interface, ...) the designers have of the targeted system, but, in addition, the experimental data produced during the experiment can serve, if thoroughly recorded, as initial and strongly relevant training data for the system.

As far as concrete implementation is concerned, an interesting and practical solution is to aim at directly integrating the WoZ capabilities in the dialogue development environment, i.e. in the dialogue interpreter and/or the dialogue manager. However, in order to do so, it is necessary to precisely identify the different entry points into the system that should be effectively provided to the wizard for potential intervention in system behaviour.

In the most general case, this might be an extremely difficult issue, as, in theory, the different dialogue models that can be considered might be of arbitrarily large complexity. In the case of the InfoVox project however, the dialogue model was sufficiently simple to allow an alternative approach that did not require modifying neither the dialogue interpreter, nor the dialogue manager. In fact, the dialogue model was completely re-implemented as a set of interconnected HTML forms, each of these forms representing one of the GDNs in the model and the branching being directly implemented in the form of hyperlinks between the HTML forms.

Each of the forms provided the wizard with the necessary functionalities to operate the simulation (the possibility to play pre-recorded messages containing the various prompts, the possibility to store and visualise the various attribute values progressively provided by a user during his/her interaction, the possibility to decide and store what the wizard thinks the interpretation produced by the GDN grammar should have been if such a grammar would have been available, ...). As far as the dialogue management is concerned, it fully relied on the Wizard who had to manually select the next active GDN or local processing (help. repetition,...) by clicking on the corresponding hyperlink. One of the main advantages of the selected approach was to make it possible to set-up the WoZ experiment very rapidly and at a very low implementational cost. However, the fact that the WoZ experiment was carried out in the form of a pure simulation (i.e. the WoZ environment was never connected with any of the running versions of the dialogue manager prototype) made the task of the wizard quite difficult (operating the simulation in real time appeared to be a task with a quite high cognitive overload). For this reason, in the Inspire project, we have implemented an extended WoZ interface that rely on a

tighter integration of the WoZ functionalities in the dialogue management environment. In particular, the WoZ interface now integrates the same dialogue management strategy as the one implemented in the dialogue manager itself. This allows the Wizard not to care about the dialogue flow management and to fully concentrate on the processing of the user utterances (i.e. the transcription and/or the extraction of the adequate canonical values which makes his/her task simpler. In addition, we are currently planning to modify the grammar processing component of the VoiceXML interpreter so that the HTML forms required for the WoZ experiment would be automatically and dynamically produced by the dialogue interpreter itself.

To guarantee an easy production of the extended WoZ interfaces, we have developed a WoZ Interface Generator which allows us to automatically create the WoZ Interface required for a given WoZ experiment. The WoZ Generator needs 2 types of inputs: the complete solution table and a configuration file containing the description of the GDNs.

The result (i.e. the produced WoZ interface) consists of two main components: an application independent library of HTML templates and Java Scripts common for all generated WoZ Interfaces and an application dependent component corresponding to an HTML interface which allows the Wizard to simulate the system in the WoZ experiment.

The main advantage of the WoZ Interface Generator is that it allows a very quick production of WoZ Interfaces which are simple to use and easy to modify which makes of it a very valuable tool for iterative dialogue model improvement.

5 Internal Field Test

The aim of the internal field-test is to improve the dialogue model, by for instance reformulating unclear prompts, and to validate the evaluation procedure (coherence, understandability). The test is conducted with the cooperation of "friendly" users, namely system designers, colleagues, friends and family, who do not necessarily represent the target users of the application. The test is conducted in the following way:

1. Description of the system and of the evaluation procedure (3 minutes);
2. The user is put in a specific applicative context with a scenario (3 minutes);
3. The user is connected to the system (5-10 minutes);
4. A satisfaction questionnaire is submitted to the user (10 minutes).

6 External Field Test

The aim of the external field test is to evaluate the final dialogue model according to the evaluation procedure defined during the internal field-test. The "external" adjective defines the fact that users in this case are randomly chosen among a set of relevant target users for the application.
The input to this test are:

- The answers to the satisfaction questionnaire, providing **subjective** indicators: average scores obtained dor the various closed questions;

- The log files produced by the system during the interactions with the users (duration, number of turns), providing **objective** indicators: average values measured for several system characteristics.

Three different kinds of analyses were performed on the input data:

1. **Retrospective trend analysis**: its purpose is to provide a synthetic view on the opinion of the users about the system, and therefore to identify the predominant trends;

2. **Retrospective correlation analysis**: its purpose is to identify dependencies between objective and subjective indicators;

3. **Prospective correlation analysis**: its purpose is to identify dependencies between objective and subjective indicators.

7 Conclusion

The RDPM is an efficient framework for state-based and frame-based approaches of spoken dialogue systems. The practical result shows that which simple applications (e.g. restaurant search in Infovox), we can develop an initial dialogue model in several hours. The dialogue manager, the most importance part of dialogue prototyping, covered most of dialogue management activities (i.e. branching logic, dialogue dead-end management strategy, confirmation strategy, dialogue termination strategy, incoherencies, strategy defining level of initiative, etc.).

The RDPM has been implemented in the form of an automated WoZ Interface Generator and dialogue management library, that allows creating WoZ interfaces automatically. Another important part in the methodology is the evaluation (steps 3,4,5). The design based on simulations method with WoZ experiments and two types of test (i.e. internal field test and external field test) [Rajman *et al.*, 2003].

In InfoVox project, the RPDM has been proposed, implemented and validated with very simple dialogue management and is targeted for simple finite-state based dialogue model. The results are (1) the restaurant information server prototype (consists of functional modules: the telephone interface, the speech recognizer, the dialogue manager, and the database manager), and (2) the dialogue prototyping and evaluation methodology [Rajman, 2003].

In Inspire project [3], the RDPM has been improved to support frame-based dialogue models, strategies for dialogue management are added and validated. Although, this is an ongoing project, we have a created a good dialogue model with all necessary dialogue functions.

[3] See http://www.inspire-project.org; the INSPIRE project is funded by the European FP5 IST research grant program.

References

[Bilange, 1992] E. Bilange. *Dialogue personne-machine, modélisation et réalisation informatique*. Langue, Raisonnement, Calcul, Hermès, Paris, 1992.

[Boyce and Gorin, 1996] J. Boyce, Susan and L. Gorin, Allen. User Interface Issues for Natural Spoken Dialog Systems. *AT&T Laboratories*, 1996.

[Catizone *et al.*, 2002] R. Catizone, A. Setzer, and Y. Wilks. State of the art in dialogue management. 2002.

[Churcher *et al.*, 1997] Gavin E. Churcher, Eric S. Atwell, and Clive Souter. Dialogue management systems: a survey and overview, 1997.

[Cohen, 1997] P. Cohen. Dialogue modeling in survey the state of the art in human language technology. 1997.

[Dahlbäck and Ahrenberg, 1993] Jönsson Arne Dahlbäck, Nils and Lars Ahrenberg. Wizard-of-Oz studies - why and how. *Knowledge Based Systems, vol. 6, No 4*, 1993.

[Daly-Jones *et al.*, 1999] Owen Daly-Jones, Nigel Bevan, and Cathy Thomas. *Wizard-of-Oz prototyping*. http://www.ejeisa.com/nectar/inuse/6.2/3-3.htm, 1999.

[Fraser and Gilbert, 1991] N. Fraser and N. Gilbert. Simulating Speech Systems. *Computer Speech and Language*, vol. 3-5, 1991.

[Geutner Petra and Dietrich, 2002] Steffens Frank Geutner Petra and Manstetten Dietrich. Design of the VICO Spoken Dialogue System : Evaluation of User Expectations by Wizard-of-Oz experiments. In *Proc. Third International Conference on language Resources and Evaluation (LREC2002)*, 2002.

[McTear, 2002] M. McTear. Spoken dialogue technology: Enabling the conversational user interface. *ACM Computing Survey, Volume 34, No 1*, 2002.

[Rajman *et al.*, 2003] M. Rajman, A. Rajman, F. Seydoux, and A. Trutnev. Assessing the usability of a dialogue management system designed in the framework of a rapid dialogue prototyping methodology. *First ISCA Tutorial Research Workshop on Auditory Quality of Systems, Akademie Mont-Cenis*, April 2003.

[Rajman, 2003] M. Rajman. Rapport scientifique - infovox project. 2003.