Introducing Reset Patterns: an Extension to a Rapid Dialogue Prototyping Methodology

I&C TECHNICAL REPORT IC/2004/58

Silvia Quarteroni and Martin Rajman
Artificial Intelligence Laboratory
Information and Communication Sciences Faculty
EPFL, Lausanne

July 9th, 2004

Abstract

This paper exposes the Rapid Dialogue Prototyping Methodology [1, 2, 3], a methodology allowing the easy and automatic derivation of an ad hoc dialogue management system from a specific task description. The goal of the produced manager is to provide the user with a dialogue based interface to easily perform the target task. In addition, reset patterns, an extension of the prototyping methodology allowing a more flexible interaction with the user, are proposed in order to improve the efficiency of the dialogue. Reset patterns are justified and theoretically validated by the definition of an average gain function to optimize. Two approaches to such an optimization are presented, focusing on a different aspect of the gain function. Eventually, experimental results are presented and a conclusion is drawn on the usefulness of the new feature.

Contents

1 Introduction **3**

2 The Rapid Dialogue Prototyping Methodology **3**
2.1 Producing the Task Model 4
2.2 Deriving the Initial Dialogue Model 5
2.2.1 Generic Dialogue Nodes 5
2.2.2 Global Dialogue Flow Management 5
2.3 Limitations of RDPM 6

3 Reset Patterns: methodology **6**
3.1 The gain function 7
3.1.1 Use of Reset Patterns 8
3.1.2 The average gain function $G(i_1)$ 9
3.2 Simplifying *G* 10
3.2.1 Hypotheses for $P(i_2|i_1)$ 10
3.3 Estimating the $P(i_2|i_1)$ 11
3.3.1 Action types 11
3.3.2 Action type transition probabilities 12
3.4 Extracting the reset patterns 12
3.4.1 The "brute force" approach 12
3.4.2 Dealing with large numbers of attributes 12
3.4.3 Example 13
3.5 Validating the logfile approach 13
3.5.1 Computing the scores 14
3.5.2 Comparing results 14
3.6 Application example 14
3.6.1 Computing the scores 15
3.6.2 Comparing results 16
3.6.3 Considering propagation 16
3.7 Corollary 17

4 Experimental Results **18**
4.1 The $P(i_2|i_1) = K$ and $P(i_2|i_1) = P(i_1)$ cases 18
4.2 Random probabilities ($P(i_2|i_1) = random$) 19
4.3 Further experimentation 19

5 Conclusions and Further Work **20**

1 Introduction

The Rapid Dialogue Prototyping Methodology [1, 2, 3] (or RDPM) aims at creating task-oriented dialogue models according to the following general idea: the target dialogue model is a finite-state model that can be easily and systematically derived from a frame based representation of the task. Such an approach is innovative when compared to traditional dialogue prototyping methods which start from a generic dialogue model and apply it to concrete cases; RDPM builds the dialogue manager specifically for a given task, thus ensuring that only necessary features are present and that the dialogue model is not too complex for the target task. Three projects have been carried out using this approach: InfoVox [1], consisting in the deployment of a dialogue-based vocal server providing information about the restaurants in the city of Martigny, Switzerland; secondly, the European INSPIRE project [2], aiming at a dialogue based control of various home devices within a Smart Home environment; finally the MDM project [3], aiming at a dialogue based interaction with a database containing multimodal meeting transcriptions. The InfoVox project [4] served to fully test the original version of RDPM, while the methodology is currently further investigated in the context of various extensions of RDPM.
We will first give a brief overview of the key features of RDPM and then focus on an extension, reset patterns, motivating it and showing its efficiency. Finally, a conclusion on the use of the extension is drawn.

2 The Rapid Dialogue Prototyping Methodology

The Rapid Dialogue Prototyping Methodology basically consists in five main steps:

1. *producing a task model* for the target application (see subsection 2.1);

2. *deriving an initial dialogue model* from the produced task model (see 2.2);

3. *carrying out a Wizard-of-Oz experiment*, a simulated human-machine interaction where the user is exposed to a system he believes fully automatic, while a hidden human operator is manually operating some of the system functionalities that have not yet been fully implemented. This enables developers to obtain an early evaluation of their prototype, and therefore to improve their dialogue model.

4. *carrying out an internal field test* to further refine the dialogue model, and to validate the evaluation procedure. The initial field test is conducted with

[1] jointly realized by EPFL, IDIAP, and the Swisscom and Omedia companies. This project is partly funded by a Swiss national CTI grant program.

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

[3] See http://www.im2.ch; the IM2.MDM project is partly funded by a Swiss national FNRS NCCR grant program.

the cooperation of "friendly" users, namely system designers, colleagues and friends, who are not necessarily representative of the target users of the application.

5. *carrying out an external field test* to evaluate the final dialogue model according to the evaluation procedure defined during the internal field test.

The first two steps will be described in more detail in the following subsections, while the three remaining ones are not relevant to this contribution and therefore will not be explained further.

2.1 Producing the Task Model

In RDPM, a task model is described in the form of a set of relational tables (frames), where columns are the attributes needed to identify the actions to be performed and lines, or "solutions", are the possible action candidates. 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 action corresponding to the values selected for the attributes.
For example, in the InfoVox restaurant application, the task model simply reduces to a single function `select_restaurant(Cuisine, Location, Opening_time, Opening_days, 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: `Cuisine`, `Location`, `Opening_time`, `Opening_days`, and `Price_range`, the lines of which are the various value combinations of the attributes corresponding to existing restaurants.

Table 1: Task model in the InfoVox application

`Cuisine`	`Location`	`Opening_time`	`Opening_days`	`Price_range`
Italian	City center	11:30 - 23:30	Thu-Tue	20 - 40 CHF
Chinese	West	11:00 - 00:00	Wed-Mon	10 - 20 CHF
...	...	...	...	...

In the case of more complex models consisting of several interconnected tables (for example a main table 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 into a single one.

2.2 Deriving the Initial Dialogue Model

In the RDPM approach, a dialogue model is defined as a set of interconnected Generic Dialogue Nodes (or GDNs), where each node is associated with one of the attributes in the solution table. For any given slot, the role of the associated GDN is to perform the simple interaction with the user that is required to obtain a valid value for the associated attribute.

2.2.1 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: 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 devices in a given room), and list processing GDNs (also called Dynamic GDNs) associated with *dynamic fields* (i.e. the values of which quickly change in time; i.e. the list of films that might be recordedat a given date and time).
To perform the interaction it is responsible for, each GDN contains 2 types of components: prompts and grammars. Prompts are the messages uttered by the GDN during the interaction. Several types of prompts are defined, handling all the possible interaction situations with the user, for instance requests for help or repetition concerning the last system question. In all cases, there is an upper limit to the number of consecutive times that a given GDN can be activated; if it is exceeded, control is handed back to the global dialogue manager with the appropriate error message.
The role of 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.

2.2.2 Global Dialogue Flow Management

In the architecture that we have selected for implementation, the processing of the GDNs (i.e. the actual interaction with the user) is taken in charge by a specific module called the *local dialogue manager*. In addition, a branching logic responsible for the management of the global dialogue flow needs to be specified as well: in our approach, this branching logic is hard-coded in a specific module called the *global dialogue manager*.
The Global Dialogue Flow Management 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 incoherences.

2.3 Limitations of RDPM

RDPM is a bottom-up methodology: it starts from a precise task and builds up a dialogue manager specifically for that task. This implies that no unnecessary features are produced, but on the other side that there are several limitations in the final dialogue manager.
One of these limitations is the absence of global memory in the system: as soon as an action has been performed, all request fields are reset to "zero" and there is therefore no way to exploit previous interaction with the system. This implementation choice can be justified by the will to create a "light" dialogue manager; however, the need of a more intelligent reset strategy than "total" reset is quite clear.
A typical example would be the following, where the user interacts with a smart home and searches for a film to record for the evening; in the current version of RDPM, the interaction would sound like:

- *System*: What can I do for you?
- *User*: I want to search for a movie on TV this evening.
- *System*: I have found "Star Wars" on ABC.

At this point, the task (providing information about the movie) has been performed and the dialogue memory is then reset to zero. A new interaction is instantiated, and the next prompt is:

- *System*: What else can I do for you?

If, on the basis of past interactions, the system could predict that the user will want to record the film it could memorize some of the past information and prevent the user from specifying a whole request from scratch. The system might then simply ask "Would you like to record the film on ABC tonight?". Implementing such memory based behavior in RDPM is the motivation behind the extension exposed in the following section, the reset patterns. Although there is yet no proof of the correlation between the use of reset patterns and increased user satisfaction. Our assumption is that entering a smaller number of parameters and therefore shortening the interaction time is a great advantage to the user; we plan to analyze user satisfaction during the ongoing WoZ experiments with the INSPIRE and MDM prototypes.

3 Reset Patterns: methodology

The main goal of reset patterns is enabling the system to adapt itself to the behavior of one specific user or population of users and to anticipate his/their next decisions. The idea is to develop an intelligent reset algorithm that sets the dialogue model field values to zero only if there is no way of accurately estimating their most probable values according to previous interactions with the user. Otherwise the

system should keep the most probable values according to the previous request. This probability can be formalized as:

$$P(D_i^{(t+1)} = v_h | A^{(t)}, U_k)$$

that is, the probability that, at step *t+1*, the attribute D_i in the request will have value v_h, knowing that the action A was uttered by the user U_k at step *t*.
Before introducing the algorithm computing the optimal reset pattern for each action, we will define in section 3.1 a criterion allowing us to assess the performance of reset patterns: a function called **gain** to be optimized in order to validate reset patterns.

3.1 The gain function

Let us consider the ith action in the solution table associated with the dialogue model:

v_{11}	v_{12}	...		
...				
v_{i1}	v_{i2}	...	v_{ij}	...
...				

where v_{ij} is the value of the j^{th} attribute of the i^{th} solution. If the j^{th} attribute is not relevant for the i^{th} solution, we write $v_{ij} = NREL$. We then define, for each solution i:

- $Def(i) = \{j | v_{ij} \neq NREL\}$: $Def(i)$ is the set of the attributes that are relevant for the i^{th} solution;
- $Set(i)$ is the set of the attributes for which a reset value v_j^i is defined.

The reset pattern $V(i)$ for the i^{th} solution is therefore the vector of values v_j^i defined by:

$$v_j^i = \begin{cases} v_{ij} & if j \in Set(i) \\ undefined & otherwise \end{cases}$$

In addition, we also define:

- $Cmp(Set(i)) = \{i' | \forall j \in Set(i), v_{ij} = v_{i'j}\}$: $Cmp(V(i))$ is the set of solutions compatible with the reset pattern $V(i)$; notice also that, by definition, $\forall i' \in Cmp(Set(i))$, $Set(i) \subset Def(i')$.

and, for any set of attributes $A \in Def(i)$,

- $n(i, A)$ is the average number of attribute values (also called information units hereafter) that the user has to provide to the dialogue system in order to set all the elements of A to values compatible with i, provided that all the attributes in $(Def(i) \setminus A)$ have already been set to compatible values.

At this point we must consider the effects of propagation, i.e. the phenomenon where, due to the constraints imposed by the solution table, entering values for one or more attributes may directly imply that the values of some other attributes are uniquely determined. The actual number of attribute values to be acquired may therefore depend on the order in which the values are provided by the user.
If none of the values of the attributes in A can be derived by propagation, we simply have: $n(i, A) = |A|$ (where $|A|$ is the size of the set A).
In case of propagation, the computation of $n(i, A)$ is more difficult: if we make the assumption that the user has no preference on the order in which the values for the attributes in A are provided, then $n(i, A)$ is the average number of information units to be provided when computed on all the permutations of A. In any case, we have: $n(i, A) <= |A|$.

3.1.1 Use of Reset Patterns

Given these definitions, the use of reset patterns is the following:

1. as soon as the action associated with solution i_1 is performed, the dialogue is restarted with all attributes in $Set(i_1)$ set to the values defined by the reset pattern $V(i_1)$ and the next prompt is produced accordingly;

2. (a) if the next solution i_2 sought by the user is in $Cmp(i_1)$, then the dialogue continues as if the values for the attributes in $Set(i_1)$ had been normally obtained from the user. The number of information units to be provided by the user in order to perform the action associated with solution i_2 can therefore be computed in the following way:
   - if no reset pattern is used, the average number of information units to provide would be: $n(i_2, Def(i_2))$;
   - if the reset pattern $V(i_1)$ is used, the dialogue starts with the attributes in $Set(i_1)$ already set to their correct values; the average number of information units to be provided is

$$n(i_2, Def(i_2) \setminus Set(i_1)).$$

   Consequently the gain in terms of information units is:

$$g(i_2|Set(i_1)) = n(i_2, Def(i_2)) - n(i_2, Def(i_2) \setminus Set(i_1)).$$

   (b) if the next solution i_2 is not in $Cmp(Set(i_1))$, then the user has to trigger the "reset" action in order to clear the dialogue memory from all reset values; the user therefore has to provide one more information unit than in the case of a dialogue without any reset pattern; in this case, the gain is therefore

$$g(i_2|Set(i_1)) = -1.$$

Let us define, for each action A_j, n_j^i as the number of values to enter to request A_j assuming that A_i is the action for which the optimal reset value (i.e. $(Set(i))$) is computed; this number is determined by:

$$n_j^i = \begin{cases} m_j^i & \text{if user agrees with pattern} \\ n_j + 1 & \text{if user doesn't agree} \end{cases} \quad (1)$$

where

- m_j^i is simplified notation for $n(j, Def(j) \setminus Set(i))$, i.e. the average number of information units to be entered to complete action A_j when the reset pattern associated to action A_i has been applied. Hence, if there is no propagation, $m_j^i = |Def(i)| - |Set(i)|$.
- n_j is the average number of information units which are necessary to identify A_j. This value corresponds to $n(j, Def(j))$.

Notice that the gain can be written as:

$$g(j|Set(i)) = n_j - n_j^i.$$

3.1.2 The average gain function $G(i_1)$

If we denote by $P(i_2|i_1)$ the probability that the action associated with i_2 is performed once the action associated with i_1 has been performed, the average gain $G(i_1)$, computed over all the possible actions, is defined by:

$$G(i_1) = \sum_{i_2} (P(i_2|i_1) * g(i_2|Set(i_1)))$$

$$= \sum_{i_2 \in Cmp(Set(i_1))} (P(i_2|i_1) * g(i_2|Set(i_1))) - \sum_{i_2 \notin Cmp(Set(i_1))} (P(i_2|i_1))$$

$$= \sum_{i_2 \in Cmp(Set(i_1))} (P(i_2|i_1) * g(i_2|Set(i_1))) - (\sum_{i_2 = 1 \to n} (P(i_2|i_1) - \sum_{i_2 \in Cmp(Set(i_1))} (P(i_2|i_1)))$$

$$= (\sum_{i_2 \in Cmp(Set(i_1))} (P(i_2|i_1) * (g(i_2|Set(i_1)) + 1))) - 1.$$

Once $G(i_1)$ is formally defined, we can use it as a quality measure for the selection of the optimal reset patterns; indeed, it seems quite intuitive to consider that the best reset patterns to choose for the action A_{i_1} is the one that maximizes $G(i_1)$, i.e. the one that, in average, minimizes the number of information units to be provided to perform the action. Notice that, as already mentioned in section 2.3, minimizing the average number of information units does not necessarily guarantee

that the the use of reset patterns will systematically lead to an increase of the user satisfaction. However, as we have already observed (in the InfoVox and INSPIRE projects), a reduction of the interaction duration often implies an increased user satisfaction. Of course, this hypothesis will be experimentally validated through specific WoZ experiments.
In this section, we present two approaches to implement the maximization of $G(i_1)$; in the first one (section 3.2), we make strong assumptions to simplify the estimation of $P(i_2|i_1)$, while in the second approach (section 3.3) we adopt a more sophisticated technique for estimating $P(i_2|i_1)$ in the case where interaction log-files are available.

3.2 Simplifying *G*

Maximizing $G(i_1)$ is obviously equivalent to maximizing, for each i_1,

$$A(i_1) = \sum_{i_2 \in Cmp(Set(i_1))} (P(i_2|i_1) * (g(i_2|Set(i_1)) + 1)).$$

In order to derive an optimal reset pattern definition, we need to compute $A(i_1)$ in a more precise way. For this, additional assumptions need to be made for the computation of $g(i_2|Set(i_1))$ and $P(i_2|i_1)$. First of all, we have:

$$g(i_2|Set(i_1)) = n(i_2, Def(i_2)) - n(i_2, Def(i_2) \setminus Set(i_1))$$

If $Set(i_1)$ is not in $Def(i_2)$, then $g(i_2|Set(i_1)) = 0$; this means that:

$$g(i_2|Set(i_1)) = g(i_2|Set(i_1) \cap Def(i_2))$$

and we can write:

$$g(i_2|Set(i_1)) = n(i_2, Def(i_2)) - n(i_2, Def(i_2) \cap Set(i_1)).$$

If we assume that there is no propagation, we have:

$$g(i_2|Set(i_1)) = |Set(i_1)|.$$

Therefore, if we consider a given reset pattern V and want to maximize $A(V)$, we have to maximize the following function:

$$A(V) = \sum_{i_2 \in Cmp(V)} P(i_2|i_1) * (|Def(i_2) \setminus V| + 1).$$

3.2.1 Hypotheses for $P(i_2|i_1)$

We can now consider three hypotheses for $P(i_2|i_1)$:

1. $P(i_2|i_1) = K$, i.e. all probabilities are the same;

2. $P(i_2|i_1) = P(i_2)$, i.e. we are only interested in the frequency of occurrence of each action without caring about sequencing;

3. the sequence between i_1 and i_2 is considered and no simplification is made.

If we consider the first two hypotheses, we can easily see that the computation of $A(V)$ will lead to a value of V which is completely independent from i_1.
As a matter of fact, in the case of the first hypothesis, we have :

$$A(V) = K * \sum_{i_2 \in Cmp(V)} (|Def(i_2) \setminus V| + 1)$$

and in the case where $P(i_2|i_1) = P(i_2)$, we have:

$$A(V) = \sum_{i_2 \in Cmp(V)} P(i_2) * (|Def(i_2) \setminus V| + 1).$$

This implies that the result will be a single optimal reset pattern optimal for all the actions. Moreover, the most compatible reset pattern reflects the most frequently occurring attribute values. This has been confirmed by experimental data as explained in section 4.
This means that ignoring the dependencies between subsequent actions is a too strong assumption and that such sequences should be considered if we want to compute actually useful reset patterns.

3.3 Estimating the $P(i_2|i_1)$

3.3.1 Action types

If we want to take into account the experience gathered from the previous interactions with the user, we can elaborate a more complex approach to the estimation of $P(i_2|i_1)$. Interaction logfiles can be used as a source for extracting the frequency of transition from an action to another; a possible solution is to store a large amount of interactions in order to derive reliable values for transition probabilities.
However, deriving transition probablility estimates from these transition frequencies may not be an easy task, as, for complex applications, the number of actions could become large and the number of observed occurrences very low. A possible solution is to cluster actions into classes A_i (hereafter called "action types") characterized by simplified action descriptions reduced to a subset of the original attributes and associated values, and to compute transition probabilities only for such action types. For instance, a possible action type in the INSPIRE Smart Home could be a two-column entry, like:

A_i	operation	device
A_0=TURNON_TV	turn on	TV

where $device$ identifies the device on which the command cmd should be performed.

3.3.2 Action type transition probabilities

One possibility for storing the transition probabilities could be a transition matrix, that is, a matrix $\mathbf{A}$ where a_{jl} represents the probability that action type A_j is followed by action type A_l. Such matrix should be computed every time the system is restarted, once a day for instance, so that the values for the transition probabilities are up to date. The reset pattern algorithm, after identifying the action type A_j of the request issued at time t, would read on line a_j in the matrix the values a_{ji}, which represent the transition probabilities to the values of action type $A^{(t+1)}$ and hence extract the transition probability values.

3.4 Extracting the reset patterns

3.4.1 The "brute force" approach

Once these transition probabilities have been estimated, we then have to elaborate an algorithm to extract the optimal reset pattern for any given action. A first possible approach is to directly use the notion of gain as defined in section 3.1. For this, we used a "brute force" approach consisting in computing, for each action (or action type), the average gain corresponding to each of the possible reset patterns, so to select the one corresponding to the maximal gain value. Results for this approach have been computed for the INSPIRE system and are provided in section 4. However, as the computational complexity of this procedure grows exponentially with the number of attributes present in the solution table, it is not realistic for large numbers of attributes.
A first option to solve this problem is to restrain to action types instead of actions, thus reducing the number of attributes to be taken into account.
To deal with the cases where it is not possible nor desirable to reduce the number of considered attributes to a tractable value, we have also developed a second solution that does not use the notion of gain. This solution is briefly presented in the rest of this section.

3.4.2 Dealing with large numbers of attributes

The idea here is to use the interaction logfiles to compute the transition probabilities between action types and attribute-value pairs. The aim is to build the reset pattern for action type A_i from the attribute-value pairs that have a probability of occurrence after A_i higher than a predefined threshold number.
If we establish a threshold for the minimal acceptable probability for an attribute-value pair, we will obtain a group of n tuples of the form $(attribute, value, probability)$, such that $probability \geq threshold$, among which to choose the values to be inserted in the target reset pattern. If no tuple exceeds the threshold, the reset pattern corresponding to the reset of all attributes is simply used; otherwise, we must determine whether the selected attribute values such values can be used to build a reset pattern: we must therefore verify that there exists an action type with which

the attribute-value pairs are compatible.
If such action type exists, then the values can be used; otherwise, we must group the n attribute-value pairs in subsets of size $n-1$ and perform the same test; if there is only one possible action type compatible with all the pairs, then those values will be used to build the next pattern; if there are more than one, the most eligible set is the one where the product of transition probabilities is the highest. If there is no compatible action, the same procedure must be applied on subsets of $n-2$ pairs, until the compatibility condition is met.

3.4.3 Example

Let us suppose that, after filtering with a threshold, the list of action types for a given application is:

Action ID	Action name
A_0	TURNON_TV
A_1	TURNOFF_TV
A_2	RAISE_SHADES
A_3	LOWER_SHADES

We suppose that after filtering with a threshold value of 0.3, we obtain the following attribute-value tuples for a given action type, e.g. A_0:

attribute	value	probability
D_0	turn off	0.7
D_0	lower	0.3
D_1	TV	0.4
D_1	shades	0.3

Since the list contains only two different attribute types (D_0 and D_1), we can directly examine the subsets made of two attribute-value pairs:

set	(attribute, value, probability)	prob. product
s1	$\{(D_0, lower, 0.3), (D_1, TV, 0.4)\}$	0.12
s2	$\{(D_0, lower, 0.3), (D_1, shades, 0.3)\}$	0.09
s3	$\{(D_0, turnoff, 0.7), (D_1, shades, 0.3)\}$	0.21
s4	$\{(D_0, turnoff, 0.7), (D_1, TV, 0.4)\}$	0.28

Since we can only retain subsets that are compatible with actual action types, we can only choose between s_2 and s_4. Since s_4 has the highest probability product, s_4 is therefore selected as the reset pattern.

3.5 Validating the logfile approach

In order to validate reset patterns computed according to the logfile probabilities, we can compute the gain g, i.e. the difference between the number of information units to be entered by the user for a given action (hereafter called *score*) in absence and in presence of reset patterns.

3.5.1 Computing the scores

First of all, we will assume that all actions in the action table are equiprobable, as there is no need to complicate the validation by the opposite assumption; the distribution of request units is also assumed to be equiprobable[4]. We will first compute, for each action A_i, the score, i.e. the average number of request units to identify A_i, when reset patterns are not active. In this case the score corresponds to $n(i, Def(i))$, which has been simplified to n_i.
Once we have obtained the values n_i, we must compute the optimal reset pattern for each action and then consider the average number of request units required for each action when reset patterns are active.

3.5.2 Comparing results

We will now consider the expected gain when one action A_i is performed, which we define as $E_i(g(j|Set(i)))$. The purpose is to compare the values for the expected gain in the case when reset patterns are active and when they are not. Obviously, the larger is the difference, the more profitable is the use of reset patterns for the specific case we are dealing with. We can say that:

$$E_i(g(j|Set(i))) = E_i(n_j - n_j^i) = n_j - E_i(n_j^i)$$

since n_j does not depend on i. Now, we can easily express $E_i(n_j^i)$, the expected number of units to be entered by the user when action A_i is followed by action A_j, by:

$$E_i(n_j^i) = \sum_i P(j|i) \cdot n_j^i.$$

In this formula, the $P(j|i)$ probability can be evaluated as the ratio between the product π_i^* of tuple probabilities for the reset pattern and the sum of products π_k of the probabilities of the tuples corresponding to incompatible actions, regardless of the threshold.
We can conclude that the expected gain can be written as[5]:

$$E_i(g(j|Set(i))) = n_j - \sum_i P(j|i) \cdot n_j^i.$$

3.6 Application example

To validate reset patterns, we will make an example based on a simplified version of the INSPIRE application. The action table is the following:

[4]Notice that assuming that action distributions are equiprobable does not imply that probabilities of transitions between actions are equivalent.

[5]Notice that, if $P(j|i) = 1/N$, $E_i(g(j|Set(i))) = n_j - 1/N \cdot \sum n_j$.

ActionID	D_0	D_1	D_2	D_3
A_0	turn on	light	left	-
A_1	turn off	light	left	-
A_2	search	TV	-	today
A_3	record	VCR	-	today
A_4	lower	shades	left	-

3.6.1 Computing the scores

We can start by computing n_i, i.e. the average number of request units to provide to identify each action A_i: when propagation isn't taken into account and no reset pattern is active (i.e. every field is reset to "zero"), such value is trivially 3.
Let us now discuss the case where reset patterns are active. As said before, we consider the logfile storing the sequence of actions performed by a single user in a given amount of time and extract the relative frequencies of such actions. We then obtain the transition matrix, which we suppose to be the following:

$$\mathbf{A} = \begin{pmatrix} 0 & 0.25 & 0.25 & 0 & 0.5 \\ 0.25 & 0 & 0.5 & 0 & 0.25 \\ 0 & 0.3 & 0 & 0.4 & 0.3 \\ 0.33 & 0.33 & 0 & 0 & 0.33 \\ 0.4 & 0.3 & 0.1 & 0 & 0.2 \end{pmatrix}$$

According to the algorithm in 3.4.1, where a threshold probability of 0.3 has been taken for each attribute-value pair, the reset patterns obtained for each action are:

A_i	V_i	π_i^*
A_0	D_0=lower (0.5) , D_1= shades (0.5), D_2=left (0.5)	0.125
A_1	D_0=search (0.5), D_1= TV (0.5), D_3=today (0.5)	0.125
A_2	D_0=record (0.4), D_1= VCR (0.4), D_3=today (0.4)	0.064
A_3	D_0=turn on (0.33), D_1= light (0.66), D_2=left (1)	0.22
A_4	D_0=turn on (0.4), D_1= light (0.7), D_2=left (0.9)	0.252

The values between parentheses are the probabilities associated to the preceding attributes and values.
If we apply formula (1) to compute the number of request units to provide in our example, we obtain the following values (the number in line i and column j in the table is the number of values to enter to request A_j assuming that A_i is the action for which the optimal reset value is computed):

	A_0	A_1	A_2	A_3	A_4
A_0	4	4	4	4	0
A_1	4	4	0	4	4
A_2	4	4	4	0	4
A_3	0	4	4	4	4
A_4	0	4	4	4	4

Notice that, as we assume that action descriptors are made of three attribute-value pairs, if one action type is compatible with a reset pattern, no more values must be provided by the user; hence, $m_j^i = 0$.

3.6.2 Comparing results

To compute the expected number of units $E_i(n_i^j)$ to enter, we must now evaluate the probabilities for the different actions. Let us take action A_0. If we consider all transition probabilities, regardless of the threshold, we have the following options:

A_1		A_2		A_4	
turn off	0.25	search	0.25	lower	0.5
light	0.25	TV	0.25	shades	0.5
left	0.75	today	0.25	left	0.75
π_1=	0.047	$\pi_2 =$	0.016	$\pi_4 =$	0.187

the optimal reset pattern being the one compatible with action A_4. This leads us to a computation of:

$$P(4|0) = \frac{\pi_4}{\pi_1 + \pi_2 + \pi_4} = 0.75.$$

This means that $E_0(n_0^4) = 0.25 \cdot 4 + 0.75 \cdot 0 = 1$. Applying the same procedure, we obtain the following values:

A_i	E_i
A_0	$0.25 \cdot 4 + 0.75 \cdot 0 = 1$
A_1	$0.33 \cdot 4 + 0.67 \cdot 0 = 1.32$
A_2	$0.63 \cdot 4 + 0.37 \cdot 0 = 2.52$
A_3	$0.6 \cdot 4 + 0.4 \cdot 0 = 2.4$
A_4	$0.535 \cdot 4 + 0.465 \cdot 0 = 2.14$

The total sum is 9.38, giving an average of 1.876 units to enter against 3 in the case without reset patterns. This implies a gain of 1.124 and proves that reset patterns give better results than the usual reset strategy.

3.6.3 Considering propagation

If we consider the previous action table description, we can notice that dependence relations can be drawn on the attributes, based on the entries in the solution table; for instance, $D_0 \Rightarrow D_1$. Therefore, the number of request units to be entered by the user varies according to the order in which such attributes are entered.
For instance, if the user enters D_0= "turn on", he will not need to enter D_1= "light", since this value always appears with value "turn on" for D_0 and therefore it will be propagated by the branching logic; however, if the user enters D_1= "light", the system will not be able to propagate a unique value for D_0.

The following table shows, for each action A_i, which sets of subsequent values are sufficient to identify such action description, and the average size of such sets, i.e. the average number of request units to provide:

A_i	possible sets of request units	n_i
A_0	(turn on), (light, turn on), (left, turn on), (left, light, turn on)	2
A_1	(turn off), (light, turn off), (left, turn off), (left, light, turn off)	2
A_2	(search), (TV), (today, TV), (today, search)	1.5
A_3	(record), (VCR), (today, record), (today, VCR)	1.5
A_4	(lower), (shades), (left, shades), (left, lower)	1.5

This gives a total sum of 8.5, and an average of 1.7 request units to enter.
If we apply formula (1) to compute the number of request units to enter in our example, we obtain the following values:

	A_0	A_1	A_2	A_3	A_4
A_0	3	3	3	3	0
A_1	3	3	0	3	3
A_2	2.5	2.5	2.5	0	2.5
A_3	0	2.5	2.5	2.5	2.5
A_4	0	2.5	2.5	2.5	2.5

Moreover, computing the E_i for our actions gives us the following results:

A_i	E_i
A_0	$0.25 \cdot 3 + 0.75 \cdot 0 = 0.75$
A_1	$0.33 \cdot 3 + 0.67 \cdot 0 = 1$
A_2	$0.63 \cdot 2.5 + 0.37 \cdot 0 = 1.575$
A_3	$0.6 \cdot 2.5 + 0.4 \cdot 0 = 1.5$
A_4	$0.535 \cdot 2.5 + 0.465 \cdot 0 = 1.3375$

The total sum is 6.1625 and the average is of 1.2325, which compared to the case where reset patterns are not used (the average number of request units was 1.7) yields a gain of 0.4675. We can thus state the eligibility of reset patterns even when constraints are considered.

3.7 Corollary

Let us now present a more general criterion to determine when the gain is positive and therefore when it is profitable to employ reset patterns. We will start from a trivial remark, which is that each probability value π_i^* associated to a chosen reset

field must be greater or equal to the remaining (not selected) values. This means that in the worst case, when a random choice is performed, $\pi_i^* = 1/N$, where N is the total number of units to be entered. Therefore the probability that the reset value does not correspond to the user's intentions is $1 - 1/N$. If we consider the "worst case scenario", i.e. the one where all reset values have been chosen arbitrarily between equiprobable values, we must solve the following equation in order to detect what is the average value of n_i that justifies reset patterns:

$$n_i = (1 - 1/N) * (n_i + 1) \Rightarrow n_i = N - 1.$$

The value for n_i must therefore be greater than or equal to $N - 1$ in order to make reset patterns useful. For instance, if the user needs to enter two values on average to identify an action, N is 3, which means that the average probability of reset patterns must be greater or equal to $1/N = 0.33$.

4 Experimental Results

We have applied the techniques exposed in this paper to the SmartHome dialogue system developed in the INSPIRE project. The complete solution table (containing 310 different actions, each defined by up to 10 attributes) was used to for compute the optimal reset patterns. For the estimation of the probabilities $P(i_2|i_1)$, three cases were considered:

- equiprobability (i.e. $P(i_2|i_1) = K$);
- no sequential dependencies (i.e. $P(i_2|i_1) = P(i_1)$);
- random probabilities.

The third case was considered to simulate a less artificial situation than cases 1 and 2. It was unfortunately impossible to use real values for the probability estimates because the INSPIRE interaction logfiles were not yet available.
For tractability reasons, we reduced the computation to a maximal number of attributes ranging from 3 to 7 and did not consider the effects of propagation.

4.1 The $P(i_2|i_1) = K$ and $P(i_2|i_1) = P(i_1)$ cases

In both cases, as predicted by the theory, there was a single optimal reset pattern, identical for all the actions. This pattern reflected the most frequent combination of attribute values, television ("Fernseher") for the *device* attribute and "action on shows" ("Sendung") for the *operation* attribute. The following table shows an excerpt from the INSPIRE solution table and indicates the unique optimal reset pattern S valid for all the actions.

Action	Device	Location	Operation	Day	Time	TVOperation
i_0	Ventilator	NREL	Zustand	NREL	NREL	NREL
i_{24}	Lampe	alle	herunter	NREL	NREL	NREL
i_{26}	Fernseher	NREL	Sendung	heute	abend	Details
S	Fernseher	-	Sendung	-	-	-

4.2 Random probabilities ($P(i_2|i_1) = random$)

Although we used random probabilities for action transitions, we tried to verify the correct behavior of the reset pattern computation algortithm by artificially strongly increasing the value of two transition probabilities:

1. the transition probability that the action "provide details on the evening film" (corresponding to the solution entry i_{45}:[Fernseher NREL Sendung heute abend Details]) occurs after action "record the evening film" (represented by i_{39}:[Fernseher NREL Sendung heute abend aufnehmen]);

2. the transition probability that the action "answering machine status" (corresponding to the solution entry i_{65}:[Anrufbeantworter NREL aktuelle_nachricht NREL NREL NREL]) occurs after the action "play the next message" (represented by i_{67}:[Anrufbeantworter NREL naechste NREL NREL NREL]).

We then computed the optimal reset patterns and obtained the following results:

Action	Device	Location	Operation	Day	Time	TVOperation
i_{45}	Fernseher	NREL	Sendung	heute	abend	Details
$S(i_{45})$	Fernseher	-	Sendung	heute	abend	aufnehmen
i_{65}	Anrufbeantworter	NREL	aktuelle_nachricht	NREL	NREL	NREL
$S(i_{65})$	Anrufbeantworter	-	naechste	-	-	-

which are precisely what we hoped to obtain. The results are therefore very encouraging and we plan to confirm them by testing the algorithm with the transition probabilities estimated from the real log files.

4.3 Further experimentation

Our further experiments should evolve along the following lines:

- computing reset patterns in the case of propagation;

- optimizing the code implementing the reset algorithm in order to allow the processing of more complex solution tables, involving a higher number of attributes;

- processing transition probabilities deriving from real interactions and evaluate if the reset patterns evaluated on that basis are pertinent to the user during the future WoZ experiments planned in the INSPIRE project.

5 Conclusions and Further Work

This paper presents an extension to the current version of our rapid dialogue prototyping methodology which enables the dialogue manager to appropriately reset part of the values of the next user request. Thus, the user is prevented from entering a new request from scratch.
First, an average gain function $G(i_1)$ is introduced to define a general criterion to maximize in order to produce the optimal reset pattern for any given action. The gain function depends on two factors: g, the difference between the number of attribute values to enter to perform an action when reset patterns are not or are active, and the action transition probabilities P.
Two approaches have been considered to perform the maximization; the first one simplifies the estimation of P; the second one proposes a more sophisticated estimation of P.The first approach proved to only produce a single optimal reset pattern identical for all of the actions. The second approach, assuming the extraction of probability estimates from interaction logfiles, was found to be much more promising.
The experiments which we conducted using the actual INSPIRE SmartHome dialogue system have shown very encouraging results; in particular, they provided interesting experimental evidence for the fact that reset patterns can indeed help the user by preventing him to provide a significant number of information units during his interaction with the dialogue system.
We are currently gathering real interaction logfiles in the framework of the INSPIRE project and plan to exploit them to validate appropriateness of the presented reset pattern strategy, especially to compute the correlation between the use of reset patterns and user satisfaction.

References

[1] 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," in *First ISCA Tutorial & Research Workshop on Auditory Quality of Systems*, Akademie Mont-Cenis, 23-25 April 2003.

[2] M. Rajman, T.H. Bui, A. Rajman, F. Seydoux, and S. Quarteroni, "Assessing the usability of a dialogue management system designed in the framework of a rapid dialogue prototyping methodology," *Acta Acustica united with Acustica, to appear*, 2004.

[3] Trung H. Bui and Martin Rajman, "Rapid Dialogue Prototyping Methodology," Technical Report No. 200401 IC/2004/01, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), January 2004.

[4] R. van Kommer, M. Rajman, and H. Bourlard, "Heading towards virtual-commerce portals," *Comtec, 9:10-13*, 2000.