Strategies for Accelerating the Design of Dialogue Applications using Heuristic Information from the Backend Database

L. F. D'Haro[1], R. Cordoba[1], R. San-Segundo[1], J. Macias-Guarasa[2], J. M. Pardo[1]

[1] Speech Technology Group - Dept. of Electronic Engineering. Universidad Politécnica de Madrid
[2] Department of Electronics. University of Alcala (UAH)

{lfdharo, cordoba, lapiz, pardo}@die.upm.es, macias@depeca.uah.es

Abstract

Nowadays, current commercial and academic platforms for developing spoken dialogue applications lack of acceleration strategies based on using heuristic information from the contents or structure of the backend database in order to speed up the definition of the dialogue flow. In this paper we describe our attempts to take advantage of these information sources using the following strategies: the quick creation of classes and attributes to define the data model structure, the semi-automatic generation and debugging of database access functions, the automatic proposal of the slots that should be preferably requested using mixed-initiative forms or the slots that are better to request one by one using directed forms, and the generation of automatic state proposals to specify the transition network that defines the dialogue flow. Subjective and objective evaluations confirm the advantages of using the proposed strategies to simplify the design, and the high acceptance of the platform and its acceleration strategies.

Index Terms: Development Platforms, Automatic Design of Dialogue Systems, Data Mining.

1. Introduction

Currently, the growing demand of automatic dialogue services for different domains, user profiles, and languages, has led to the development of many commercial and academic platforms that provide all the necessary components for designing, executing and maintaining such services with a minimum effort and with innovative functions that make them interesting for developers and final users.

In an effort for accelerating the design of multimodal and multilingual dialogue applications, all commercial platforms support dedicated hardware and state-of-the-art modules such as language identification, speech recognizers and synthesizers, etc., optimized to guarantee users satisfaction and minimum fine-tuning in the run-time system. The use of user-friendly graphical interfaces simplifies the development of complex dialogues, together with the inclusion of built-in libraries for typical dialogue states such as requesting card or social security numbers, etc., and additional assistants for debugging, logging, simulating and deploying the service. Finally, the generation of the runtime scripts using widespread standards such as VoiceXML, SALT, CCXML, etc., increases the portability and reduces costs.

In contrast to commercial platforms, academic platforms (e.g. CSLU-RAD[1], DialogDesigner[2], Olympus[3], etc.) do not necessarily incorporate all the features mentioned above, but allow more complex dialogue interactions. Most of them are freely available as open source, and their functionalities can be extended using third party modules.

Nowadays, just a few research platforms include some kind of acceleration strategies to the design based on the contents or the structure of the database. We could mention the following ones.

In [9], different data mining techniques are used to automate the selection of content data to be used in system initiative queries, and to provide summarized answers. At runtime, the system dynamically selects those that best narrow down the interaction flow with the final users.

In [8], a complete platform to build voice apps is described. Here, the system uses the dynamic contents of the database to create new grammars and prompts, as well as the dialogue flow for presenting information to the user, or for solving errors, through predefined templates and user profiles.

Finally, in [4] the contents of corporate websites are used to create automatically spoken and text-based dialogue applications. Although the dialogue flow is predefined, this paper shows that important knowledge can be extracted from a well-designed content, and how it can be incorporated into the different modules of the dialogue system.

The paper is organized as follows: section 2 provides an overview of the overall platform architecture. Section 3 describes the proposed acceleration strategies in detail. Section 4 describes the subjective and objective evaluations, and section 5 outlines conclusions and future work.

2. Platform Architecture

This paper is a continuation of the work done in the European project Gemini (IST-2001-32343), where the objective was to create an open and modular platform for the development of user-friendly, natural, multilingual and multi-modal dialogue applications, called the Application Generation Platform (AGP), which is made up of different assistants and tools. It consists of three main layers integrated into a common graphical user interface (GUI) that guides the designer step-by-step. In the first one, the framework layer, the designer specifies global aspects related to the application and the data. This layer includes the Data Model Assistant (DMA), where the database structure is created, and the Data Connector Model Assistant (DCMA), where the application specific database access functions are created. The second layer, called retrieval layer, is modality and language independent. This layer includes the State Flow Model Assistant (SFMA) and the Retrieval Model Assistant (RMA). The designer first uses the SFMA to create the dialogue flow at an abstract level, by specifying the high-level states of the dialogue, plus the slots to ask to the user and the transitions among states. Then, the RMA is used to include all the actions (e.g., loops, if-conditions, math or string operations, conditions for

[1] http://cslu.cse.ogi.edu/toolkit/
[2] http://spokendialogue.dk/
[3] http://www.ravenclaw-olympus.org/

making transitions between states, calls to dialogs to provide/obtain information to/from the user, etc.) to be done in each state defined in the SFMA. Finally, the third layer, called the dialogs layer, contains the assistants that complete the general flow specifying for each dialogue the details that are modality and language dependent. Here, for example, the prompts and grammars for each language, the appearance and contents of the Web pages, the treatment of speech recognition or Internet access errors, the presentation of information on screen or using speech, etc., are defined. Furthermore, the VoiceXML and xHTML scripts for the speech and web modalities are also automatically generated. Further details of the AGP can be found in [2] and [3].

3. Proposed Accelerations

In [2] and [3], we described our initial steps to include several acceleration strategies, based mainly on exploiting the database structure, applied successfully to different platform assistants, with a special emphasis in the RMA. In this work, we apply new strategies that exploit the database contents incorporated mainly into the Data Model Assistant, the Data Connector Model Assistant and the State Flow Model Assistant. The next sections describe in detail these assistants and the new acceleration strategies.

3.1. Strategies to the Data Model Assistant (DMA)

In this assistant, the designer creates the data model structure of the service through class descriptions. These classes provide information about which fields in the database are relevant for the service and their organization. A class can be characterized by a list of attributes, a description, and optionally a list of base classes (inheriting their attributes); and the attributes should correspond to information to be requested/presented to the user in one or more dialogue states. Attributes may be: a) of atomic types (e.g., string, Boolean, float, date, etc.), b) complex objects, obtained by embedding or referring to an existing class, or c) lists of either atomic type items or complex objects.

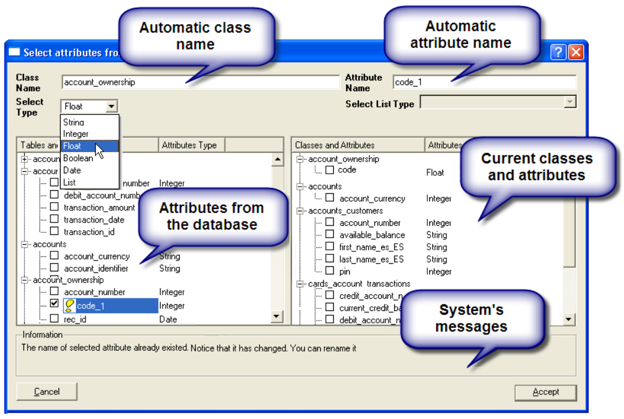
In our previous work [2][3], the assistant included the following strategies: a) support for using libraries of models, which can be copied totally or partially, or even mix several classes, b) automatic creation of a class when it is referenced as an attribute inside another one, and c) definition of classes inheriting a base class attributes.

In this work, we have added the possibility of including information regarding the relationship between class attributes and the fields and tables in the database. In order to accelerate the design, the system automatically extracts and analyzes heuristic information from the database contents and proposes full custom classes and attributes.

3.1.1. Extraction of heuristic information

This process is done using an open SQL query to retrieve information of every table, field and record in the database. This information includes the name and number of the tables and fields, and the number of records for every table. In addition, the following features for each field are also generated: a) average length, b) the proportion of records that are different, c) field type, d) number of empty records, and e) language dependent fields. These features are mainly used to simplify the design or to improve the presentation of information in the posterior assistants.

These heuristics are currently used for: (a) and (b) to unify slots as mixed initiative or not (see section 3.3.1), (c) to accelerate the creation of the data model structure (section 3.1.2), (d) to sort by relevance the attributes displayed by the wizard when creating the database structure (section 3.1.2), and (e) to avoid the proposal of states in the SFMA that will never be used (section 3.3.2) since the dialogue flow in this assistant is language independent.



Class Name: account_ownership
Attribute Name: code_1
Select Type: Float (String, Integer, Float, Boolean, Date, List)
Select List Type:

Tables and		Attributes Type
account		
account		
	number	Integer
debit_account_numb		
transaction_amount		
transaction_date		
transaction_id		
accounts		
account_currency		String
account_identifier		String
account_ownership		
account_number		Integer
☑ code_1		Integer
rec_id		Date

Classes and Attributes	Attributes
account_ownership	
code	Float
accounts	
account_currency	Integer
accounts_customers	
account_number	Integer
available_balance	String
first_name_es_ES	String
last_name_es_ES	String
pin	Integer
cards_account transactions	
credit_account_n	
current_credit_ba	
debit_account_n	

Information
The name of selected attribute already existed. Notice that it has changed. You can rename it

Figure 1. Wizard for semi-automatic class proposal.

3.1.2. Semi-automatic class proposal

After collecting all the heuristics, the assistant includes a wizard (see Figure 1) that allows the designer to create custom classes selecting the tables and fields of the database (left side in Figure 1) and/or from already existing classes in the model (right side). The heuristic information is used to set automatically the field types in the wizard. For example, in Figure 1 the field type for code in the database is String (the most generic type), but the wizard changes it to integer because all the values are actually integer, although it can be edited by the designer. Besides, the wizard also proposes automatic alternative names for the new class and attributes when it detects duplicated names with already defined ones. In the example, the system proposes code_1.

3.2. Strategies Applied to the Data Connector Model Assistant (DCMA)

This assistant allows the definition of the prototypes (i.e. only the input and output parameters) of the database access functions used in the runtime system. The advantage of using prototypes is that their actual implementation is not required during the design of the dialogue flow.

The main acceleration strategy, included in the first version of the assistant, was the association of the input/output arguments to attributes and classes defined in the data model structure (section 3.1). This information is used by the retrieval model assistant (RMA)[2] to create dialogue proposals and to automatically propose database access functions for a given dialogue in the design.

3.2.1. Semi-automatic generation of SQL queries

In this case we have incorporated a wizard that simplifies the process of creating the function prototypes (API), reducing the necessity of learning a new programming language (SQL), and that simplifies the process of adding the proposed query into the real-time modules and scripts. The new wizard semi-automatically creates the SQL statements for the given prototype and provides a pre-view of the results that the system would retrieve in the real-time system. In contrast, current platforms do not provide automatic proposals.

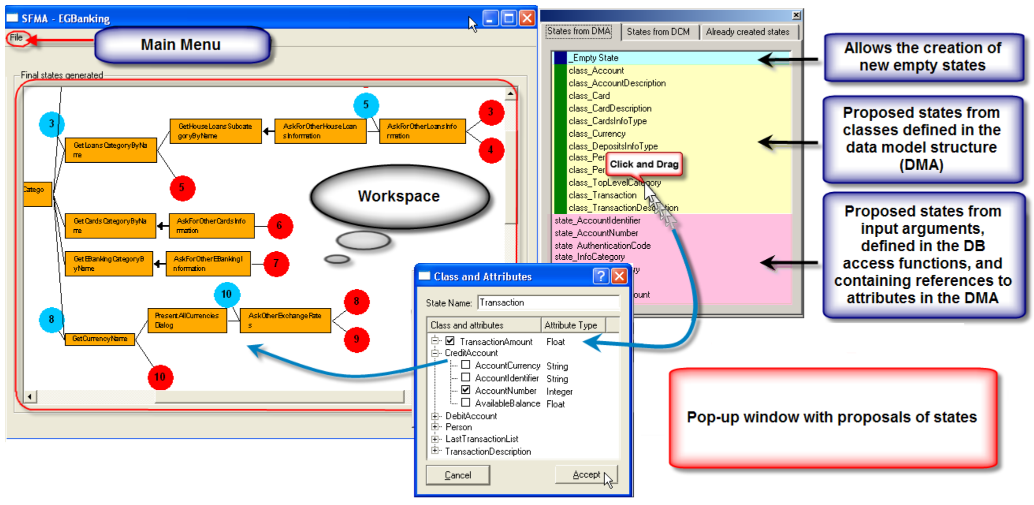


Figure 2. SFMA workspace and pop up window with state proposals from classes defined in the data model structure

The new wizard uses the information of the relationships between the arguments and the database model and database fields to automatically create the SQL statement using the input arguments of the function prototype as constraints in the WHERE clause and the output arguments as returned fields in the SELECT clause. The wizard also allows the inclusion of new input/output arguments or constraints supported by the SQL standard if the prototype is not still complete.

3.3. Strategies Applied to the State Flow Model Assistant (SFMA)

This assistant allows the designer to create a state transition network [7] that represents the dialogue flow at an abstract level, i.e. specifying only the high-level states of the dialogue, the slots to be asked to the user, and the transitions between states, but not the specific details of each state. The GUI allows the definition of new states using wizard-driven steps and a drag-and-drop interface. An important strategy from the previous version is the possibility of specifying the slots through attributes offered automatically from the data model. The new acceleration strategies are the unification of the slots to be requested using system or mixed initiative forms (section 3.3.1), and the automatic generation of state proposals (section 3.3.2).

3.3.1. Automatic unification of slots for mixed initiative

This acceleration strategy helps the designer to decide when two or more slots are good candidates to be requested one by one (using directed forms) or at the same time (using mixed-initiative forms) according to the VoiceXML terminology. This functionality is only available when the slots in a given state are all related to a table and field in the backend database (section 3.2). The assistant uses the heuristics obtained for the given fields (section 3.1.1) and applies a set of customizable rules to decide which slots can be unified and which ones cannot.

For instance, the system does not propose the unification when: a) there are two slots defined as strings and the sum of the average length of both is greater than 20 characters; in this case, the system avoids the recognition of very long sentences, b) one of the slots is defined as a string with an average length greater than 10 characters, and the other slot is an integer/float number with an average length of 5 characters. In this case, the rule avoids the recognition of long strings, e.g. an address or name, plus the recognition of long numeric quantities, e.g. phone or social security numbers, c) there are two numeric slots with a proportion of different values close to one, and the total number of records of both fields is high (configurable value), then the system determines that these slots have a large vocabulary and a high probability of misrecognition. So, in all 3 cases, the system decides that it is better to ask one slot at a time (system initiative).

3.3.2. Automatic states

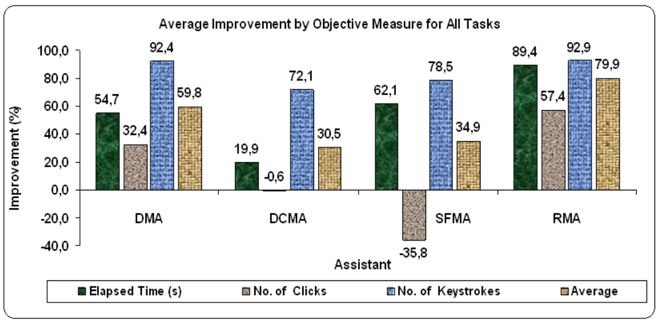
In this strategy, the assistant creates automatically dialogue states that include the slots to be requested to the user. Using the information of the database structure (DMA) and the database access functions (DCMA), the assistant creates the following kinds of state proposals.

Class dependent states: For each class defined in the DMA, the assistant creates a class template that the designer can drag and drop into the workspace. Then, a pop-up window allows the designer to select the attributes s/he wants to use as slots in the new state. Finally, the new state is inserted into the workspace allowing the designer to define the transitions (i.e. connections) to other states. Figure 2 shows an example of using the template *class_Transaction*. In this case the designer selects the attributes *TransactionAmount* and *AccountNumber* to be used as slots in the new state *Transaction*. Observe that the assistant expands complex attributes (with inheritance and objects) allowing only the selection of atomic attributes.

From database access functions: In this case, the system analyzes all the database functions defined in the DCMA containing input arguments defined as atomic types.

Then, the system uses the name of the function as proposal for the name of the state, and the input arguments as slots for that state. Again, the assistant allows the designer to select several of these proposals in order to create more complex states. For instance, if there is a database access function called *perfomTransaction*, which receives three input arguments (i.e. *DebitAccountNumber*, *CreditAccountNumber* and *TransactionAmmount*), the system automatically creates a new state proposal called *perfomTransaction* that includes the three slots. Applying similar rules to the ones described in section 3.3.1 the system would propose to request them one by one instead of using mixed-initiative.

Others: Empty states for allowing top-down design and single-slot states from the input arguments of the database access functions defined in the DCMA.



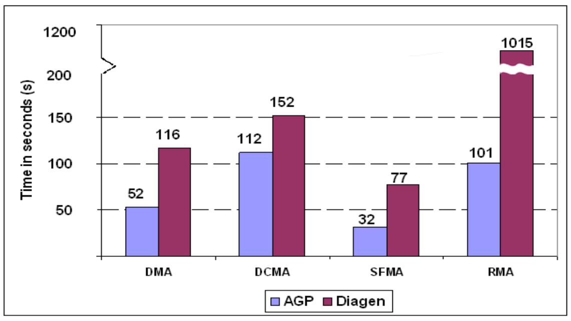


Figure 3.Objective metrics for all the evaluators/tasks: a) Average improvements, b) Elapsed time in seconds

4. Evaluation

With the objective of evaluating the performance of each of the assistants that make up the platform and the acceleration strategies described above, we carried out a subjective and objective evaluation with 9 developers with different experience levels and profiles (4 novices, 3 intermediates, and 2 experts) on designing dialogue services, where experts had previous experience with this platform and at least two others. All of them were requested to fulfil the same tasks covering each of the proposed accelerations and assistants to evaluate, e.g. to create a class model with two atomic and one complex attributes (DMA), to create a state with mixed initiative slots (SFMA), and to create a dialogue with over-answering and an IF-Then-Else condition (RMA). Further details can be obtained in [1]. Even though improvements in the RMA are described in [2] and [3], we include here its evaluation results for homogeneity and to demonstrate the effectiveness of our accelerations in the most complex assistant.

For the objective evaluation, we collected the metrics proposed in [6]: elapsed time, number of clicks, and number of keystrokes. The metrics obtained with the accelerated assistants were compared with the collected using a built-in editor called Diagen [5], which features fewer accelerations but generates the same information specified by our assistants. As accelerations, Diagen only provides default templates that the designer has to complete and a guided procedure using different pop-up windows to fulfil the templates.

The results, see Figure 3a, confirm that the design time can be reduced, in average for all the assistants, evaluators and tasks, in more than 56%, the number of keystrokes in 84%, and the number of clicks in 14%. The results in Figure 3b show the elapsed time for comparing our platform with accelerations and Diagen. It is important to highlight the important reductions, one order of magnitude, in the RMA considering that it is the main task in the design. Finally, it is also important to mention the differences between the evaluator profiles. In detail, for the AGP the average elapsed time for the experts was 25% better than for the intermediates and 53% lower than for the novices. For Diagen, the elapsed time for experts was 20% lower than for intermediates and 37% lower than for novices. These results confirm that the differences between experts and novices are reduced when both are requested to do the same task but without the accelerations.

In the subjective evaluation, the DMA and DCMA were both scored with 8.3, the SFMA with 9.0, the RMA with 8.6, and Diagen with 4.5. Regarding the acceleration strategies, the evaluators scored the automatic states with 9.3, the SQL generation and the unification of slots for MI with 9.0, and the class proposals with 8.9. These results confirm the designer-friendliness of the assistants and accelerations, as well as their usability, in contrast to Diagen.

5. Conclusions and Future Work

In this paper, we have described a set of new and innovative acceleration strategies based on using heuristic information extracted from the backend database of the service in order to accelerate the design of multimodal and multilingual dialogue apps. Our proposals include the creation of automatic state proposals, the unification of slots to be requested using mixed-initiative dialogues, and the semi-automatic creation and debugging of SQL statements. Subjective and objective evaluations confirm that the strategies are useful and contribute to simplify and accelerate the design.

As future work, we propose the creation of new rules for unifying slots for mixed-initiative dialogues, to improve the GUI to define the database access prototypes by offering more automatisms, and the extraction of new heuristics to detect automatically the relationship between tables and fields in order to propose more complex classes in the DMA.

6. Acknowledgements

This work has been supported by SD-TEAM (TIN2008-06856-C05-03/05) and ROBONAUTA (DPI2007-66846-c02-02).

7. References

[1] D'Haro. L. F. "Speed Up Strategies for the Creation of Multimodal and Multilingual Dialogue Systems". PhD Thesis. Universidad Politécnica de Madrid. 2009.
[2] D'Haro, L. F., Cordoba, et.al. "An advanced platform to speed up the design of multilingual dialogue applications for multiple modalities" Speech Communication (48):8, pp. 863-887, 2006.
[3] D'Haro, L. F., Cordoba, R., et. al., "Strategies to reduce design time in multimodal/multilingual dialog applications". ICSLP 2004, pp IV-3057–3060.
[4] Feng, J., Bangalore, S., Rahim, M. "WEBTALK: Mining Websites for Automatically Building Dialog Systems". ASRU 2003, pp. 168-173.
[5] Hamerich, S. "From GEMINI to DiaGen: Improving Development of Speech Dialogues for Embedded Systems". 9th SIGDIAL 2008, pp. 92-95.
[6] Jung, S., Lee, C., et. al. "DialogStudio : A Workbench for Data-driven Spoken Dialogue System Development and Management". Speech Communications, 50 (8-9), pp. 683-697. 2008.
[7] McTear, M. "Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit". ICSLP 1998, pp. 1223–1226.
[8] Pargellis, A. N., Kuo, H. J., Lee, C. "An automatic dialogue generation platform for personalized dialogue applications". Speech Communication Vol. 42, pp. 329-351. 2004.
[9] Polifroni, J. and Walker, M. "Learning Database Content for Spoken Dialogue System Design". LREC 2006, pp. 143-148.