Speeding Up the Design of Dialogue Applications by Using Database Contents and Structure Information

L. F. D'Haro, R. Cordoba, J. M. Lucas, R. Barra-Chicote, R. San-Segundo
Speech Technology Group
Dept. of Electronic Engineering
Universidad Politécnica de Madrid, Spain
{lfdharo, cordoba, juanmak, barra, lapiz}@die.upm.es

Abstract

Nowadays, most commercial and research dialogue applications for call centers are created using sophisticated and fully-feature development platforms. Surprisingly, most of them lack of some kind of acceleration strategy based on an automatic analysis of the contents or structure of the backend database. This paper describes our efforts to incorporate this kind of information which continues the work done in (D'Haro et al, 2006). Our main proposed strategies are: the generation of automatic state proposals for defining the dialogue flow network, the automatic selection of slots to be requested using mixed-initiative, the semi-automatic generation of SQL statements, and the quick generation of the data model of the application and the connection with the database fields. Subjective and objective evaluations demonstrate the advantages of using the accelerations and their high acceptance, both in our current proposals and in previous work.

1 Introduction

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

In their effort for accelerating the design, most commercial platforms provide several high-level tools to build multimodal and multilingual dialogue applications using widespread standards such as VoiceXML, CCXML, J2EE, RCP, SRGS, etc. In addition, they include state-of-the-art modules such as speech recognizers, high quality speech synthesizers, language identification capabilities, etc., that guarantees user satisfaction and interaction. In addition, they present a very user-friendly graphical interface that makes easy the development of very complex dialogues, besides the incorporation of predefined libraries for typical dialogues states such as requesting card or social security numbers, etc., and additional assistants for debugging, logging and simulate the service.

In contrast to commercial platforms, research or academic platforms (e.g. CSLU-RAD [1], DialogDesigner[2], Olympus[3], Trindi-kit[4], etc.) do not necessary incorporate all the above-mentioned features; especially because they are limited to the number of standards that they are able to handle and to the integration level with other platforms, as well as the number of capabilities that they can offer to the users and programmers. However, they allow more complex dialogue interactions, most of them are freely available as open source, and using third party modules it is possible to extend their functionalities.

Surprisingly, these platforms do not include any kind of acceleration strategies based on the contents or in the structure of the backend database that, as we will show, can provide important information for the design. Next, we will describe some examples of applications or dialogue systems that use data mining techniques or heuristic infor-

[1] http://cslu.cse.ogi.edu/toolkit/
[2] http://spokendialogue.dk/
[3] http://www.ravenclaw-olympus.org/
[4] http://www.ling.gu.se/projekt/trindi/trindikit/

Proceedings of SIGDIAL 2009: the 10th Annual Meeting of the Special Interest Group in Discourse and Dialogue, pages 160–169,
Queen Mary University of London, September 2009.

mation extracted from the database contents in order to create automatic dialogue services.

In (Polifroni and Walker, 2006), 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 automatically selects the attributes to constrain the prompt queries that narrow down best the interaction flow with the final users.

In (Chung, 2004), the database is used together with a simulation system in order to generate thousands of unique dialogues that can be used to train the speech recognizer and the understanding module, as well as to diagnose the system behaviour against problematic user's interactions or answers.

In (Pargellis et al, 2004), a complete platform to build voice services where the database contents change constantly is described. At runtime, the system retrieves information that the user is interested in according to his personal profile. In addition, the system is able to create automatically dynamic speech grammars and prompts, as well as the dialogue flow for presenting information to the user, or for solving some interaction errors through predefined dialogue templates.

Finally, (Feng et al, 2003) proposes a very different approach, not using a database but mining the content of corporate websites for automatically creating spoken and text-based dialogue applications for custom care. Although the dialogue flow is predefined, it is interesting to see that important knowledge, for the different modules of the dialogue system, can also be extracted and used from a well-designed content.

In this work, we have solved some of the limitations of current platforms by incorporating successfully heuristic information into the different assistants of the platform and allowing them to collaborate between each other in several ways, as they collect the information already provided in the first stages of the design to improve and accelerate the design in the last stages. This way, the platform assistants classify which fields of the database could be relevant for the design, generate different kinds of automatic proposals according to the design step, reduce the information displayed to the designer, and accelerate different typical procedures required to define the application.

The paper is organized as follows: section 2 provides an overview of the overall architecture of the platform, including a brief description of the main assistants and layers that makes it up. Section 3 describes previous accelerations in the platform related with the current work; Sections 4, 5, and 6 describe in detail the new strategies and the assistants that include them. Section 7 describes the subjective and objective evaluations, and section 8 outlines some conclusions and future work.

2 Platform Architecture

The Application Generation Platform (AGP), created during the European project GEMINI, is an open and modular architecture made up of different assistants and tools that simplifies the generation of multimodal and multilingual dialogue applications with a high adaptability to different kinds of services (see Figure 4 in Appendix A). The platform consists of three main layers integrated into a common graphical user interface (GUI) that guides the designer step-by-step and lets him go back and forth.

In the first layer, called 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 next layer, called Retrieval layer, includes the State Flow Model Assistant (SFMA) and the Retrieval Model Assistant (RMA). The former is used 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 later is used to include all the actions (e.g., variables, loops, if-conditions, math or string operations, conditions for making transitions between states, calls to dialogs to provide/obtain information to/from the user) to be done in each state defined previously.

Finally, the third layer, called Dialogs Layer, contains the assistants that complete the general flow specifying for each dialogue the details that are modality and language dependent. For instance, the prompts and grammars for each language and modality, the definition of user profiles, the appearance and contents of the Web pages, the error treatment for speech recognition errors or Internet access, the presentation of information on screen or using speech, etc., are defined. Furthermore, the

VoiceXML and xHTML scripts used by the real-time system are automatically generated.

3 Previous Acceleration Strategies

In (D'Haro et al, 2006) and (D'Haro et al, 2004), we described several acceleration strategies based on using the data model structure and applied them successfully to different assistants of the platform, with a special emphasis in the assistant for defining the actions to be done in each dialogue (i.e. RMA). The data model information was used to:

a.) Create configurable and generic dialogue proposals for obtaining (called DGet) and for showing (called DSay) information from/to the user. In this case, the assistant creates a DGet or DSay dialogue for each class and attribute defined.

b.) Automatically propose the actions required for completing the information for each state of the dialogue flow; basically, the assistant proposes the dialogues to ask information to the user, the database access functions, and the dialogues to show information to the user. Figure 1 shows an example of the proposals for a banking application. In this example, the designer is editing a dialogue where given a currency name the system provides its specific information (buy and sell price, general information, etc.). Using the proposal window, all the designer would need to do is to select the corresponding DGet in the window (*DGet_CurrencyName_IN_CLASS_Currency*), then the database access function *GetCurrencyByName*, and finally the DSays that provide the desired attributes from the currency. In order to provide these proposals, we use the information of the relationships between slots and arguments of the database functions and the attributes and classes in the data model (section 5 and 6). When there is no relationship specified, we apply relaxed filters such as matching in types, similarity in names, or same number of arguments and slots in the state.

c.) Automate the process of passing information among actions/dialogues by proposing the variables that best match the connections or allowing the creation of new variables when no match exists. This is a critical aspect of dialogue applications design. Several actions and states have to be 'connected' as they use the information from the preceding dialogues. In general, most current design platforms allow the same kind of functionality, offering the user a selectable list of all the available variables in the dialogue. In other cases, especially considering the connections with database access functions, some platforms only allow the designer to define the matching by modifying by hand the script code. In this acceleration, we have tried to provide a better solution by automating the connection through automatic proposals. The assistant detects the input/output variables required in each action and offers the most suitable already defined variable of a compatible type; if there are more than one variable to show, the assistant sorts them according to the name similarity between variable and dialogue. If there is no compatible variable already defined in the system or the name proposed is not desired, the assistant allows the creation of a new local/global variable. Additionally, the assistant includes a window where all this matching can be edited.

Other accelerations included in this assistant were the quick creation of mixed-initiative dialogues, dialogues with over-answering (that do not exist in any current dialogue platform) and the quick definition of dialogue variables.

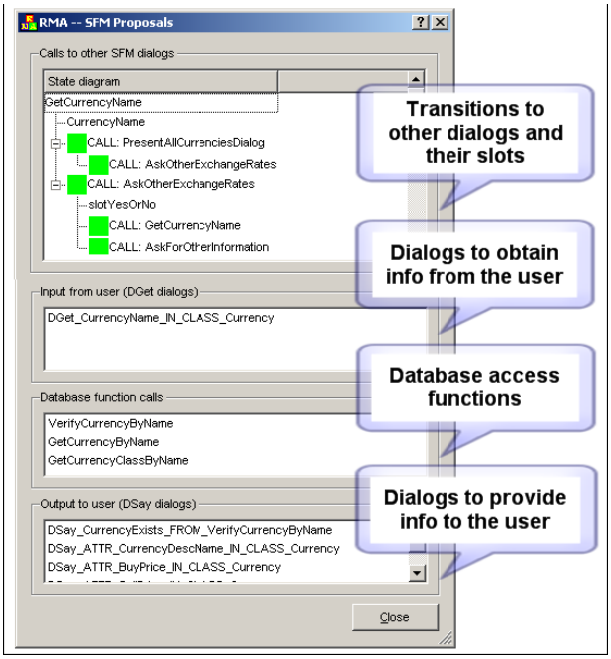


Figure 1. Example of automatic dialogues and database access function proposals

In the present work, the new accelerations additionally exploit the database contents and have been incorporated into the assistant to define the data model structure (section 4), into the assistant for defining the database access functions (section

5), and into the assistant to define the states of the dialogue flow (section 6). The next sections describe in detail these assistants and accelerations.

4 Strategies Applied to the Data Model Assistant (DMA)

This assistant helps in the creation of the data model structure of the service through a visual representation of all possible fields to be requested and presented to the user, which consists of object oriented classes and attributes. The goal with these classes and attributes is to provide information to the next assistants in the platform about which fields in the database are relevant for the service and the relationships between tables and fields.

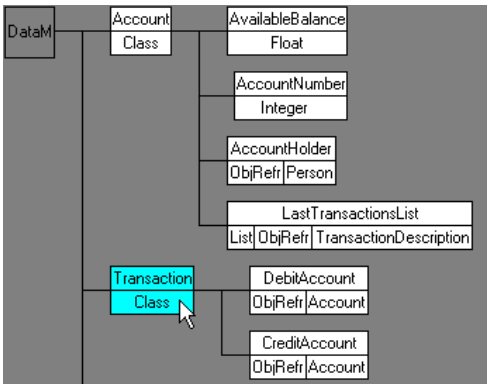


Figure 2. Example of class and attributes

Each class, see Figure 2, can be characterized by a list of attributes, a description, and optionally a list of base classes (inheriting their attributes). The attributes may be: a) of atomic types (e.g., string, Boolean, float, date, etc., e.g., *AvailableBalance*), b) complex objects, obtained by embedding or referring to an existing class (e.g., *AccountHolder*), or c) lists of either atomic type items or complex objects (e.g., *LastTransactionList*).

The main acceleration strategies, previously included in this assistant, are: a) re-utilization of libraries with models created beforehand, which can be copied totally or partially, or used to create a new class by mixing them, b) automatic creation of a class when it is referenced as an attribute inside another one, and c) definition of classes inheriting the attributes of a base class. Since this is one of the first assistants of the platform, a significant effort was done to accelerate the creation of the database structure and to include information about the relationships between the class attributes and the fields and tables in the database. To start with, the system generates and analyzes automatically heuristic information from the database contents. Then, with this information, the system proposes full custom classes and attributes that the designer can use when creating the data structure.

4.1 Extraction of heuristic information

The 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) field type, b) average length, c) number of empty records, d) language dependent fields, and e) the proportion of records that are different. This information is shared among the assistants in order to simplify the design or to improve the presentation of information in the posterior assistants. For instance, they are used for: (a) to accelerate the creation of the data model structure (section 4.2), (b) and (e) to unify slots as mixed initiative or not (see section 6.1), (c) to sort by relevance the attributes displayed by the wizard when creating the database structure (section 4.2), and (d) to not generate states for these fields in the SFMA since the dialogue flow in this assistant is language independent (section 6.1).

An important issue we found when retrieving the field type was that sometimes the metadata information provided by the SQL function was incorrect due to: a) the driver for accessing the database was only able to return a limited number of types, e.g. Boolean or dates were mapped as integer or string types respectively, b) the designer of the database defined the field using a generic type such as string or float when the visual inspection of the records showed that they actually corresponded to dates or integers, c) there were problems to map special types such as hyperlinks, binary, etc. into the types supported by our platform.

In order to solve these problems, we implemented a post-processing step based on using regular expressions to detect the following types: integers, floats, dates, strings, Boolean, mixed or empty fields. In general, the process is to analyze all non-empty records in a given field and to select as field type the one with more than the 90% of occurrences. Exceptions to this rule are: a) a numeric field is considered integer if all its records are classified as such, if not it is classified as float,

b) the empty type is assigned to fields with more than 95% of empty records.

In order to analyse the performance of the post-processing step, an objective evaluation was carried out. In this evaluation, twenty-one databases, most of them available online, were retrieved and visually inspected field by field. In total, there were 109 tables (an average of 5 tables per database), 767 fields and 610.506 records, which were classified by a human evaluator.

In our results, the average recognition was 89.6%, obtaining the best rates for dates, strings, and numeric quantities, which are the most common types in most databases. Analyzing in detail the misrecognitions, 0.9% of floats were incorrectly detected as integers due to values such as 2.0, 30.0, etc. which were automatically returned by the database driver without the decimal part. Another source of errors was detecting some numeric quantities due to special symbols such as dashes, percentages, or the euro symbol, which were incorrectly interpreted as a string type (3.3% and 1.6%). The major problems occurred for distinguishing between the String type and what we called Mixed type (i.e. fields containing: URLs, emails, long strings, etc.) since they are, in practice, the same. However, we wanted to separate them since for a speech recognizer they may be handled using different strategies (e.g. spelling, general grammars, etc.). The importance of these results is that they mean a reduction in the number of times the designer will need to change the proposed type for a given attribute when creating the classes (section 4.2).

4.2 Semi-automatic classes proposals

After collecting all the heuristics, the assistant provides a wizard window that allows the designer to create the attributes for a new class from the tables and fields of the database or from already existing classes in the model. The information of the selected field and table is saved in the definition of the class attribute allowing future assistants in the platform to access this information easily (section 5.1 and 6.1). The heuristic information is used to set automatically the field types in the wizard, 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.

Finally, if the number of tables in the database is too high the designer can select those that will be really needed during the design, this way reducing the information displayed on the screen. In addition, it is also possible to customize the name of the tables in the database in order to make them more intuitive to the dialogue designer.

5 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, previously included in this assistant, was the possibility of relating the input/output arguments to the attributes and classes of the data model. This information is used by the retrieval model assistant to create dialogue proposals and to automatically propose database access functions for a given dialogue in the design (section 3). In this work, we have introduced a new acceleration by incorporating a wizard window that allows the creation and debugging of the SQL statements used at run-time.

5.1 Semi-automatic generation of SQL queries

The main motivation behind this wizard window was to simplify the process of creating the function prototypes (API), reducing the necessity of learning a new programming language (SQL), and to simplify the process of adding the 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 (see Figure 5 in Appendix A). This new acceleration is interesting since currently few development platforms include such kind of wizard forcing the designer to use third party software. Besides, current wizards only provide debugging tools, nice GUI features or support for many DB standards, but no automatic query proposals.

In order to automatically create the SQL statement, the assistant uses the input arguments (defined in the function prototype) as constraints for the WHERE clause, and the information of the

output arguments as returned fields for the SELECT clause. The assistant allows the inclusion of new input or output arguments if the function prototype is not complete or if the designer wants to test new combinations of arguments.

Finally, the wizard allows the designer to preview the records that the proposed SQL statement will retrieve at real-time. In order to debug the query, the designer specifies, using a pop-up window, the values for the input arguments of the function to test the query (as acceleration, the wizard automatically proposes real values retrieved from the database).

6 Strategies Applied to the State Flow Model Assistant (SFMA)

This assistant is used to define 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 flow is specified using a state transition network representation that is common in this kind of platforms and dialogue modelling. The GUI allows the definition of new states using wizard-driven steps and a drag-and-drop interface. An important acceleration strategy from the previous version is the possibility of specifying the slots through attributes offered automatically from the data model. The new accelerations are the automatic proposal of the slots to be requested using system or mixed initiative dialogues (section 6.1) and the automatic generation of proposals of states for defining the dialogue flow (section 6.2).

6.1 Automatic unification of slots for mixed initiative

The idea of this acceleration is to allow the system to propose automatically when two or more slots must be requested one by one (using directed forms) or at the same time (using mixed initiative forms) according to the VoiceXML standard.

This functionality is only available when the slots to be analyzed have been defined from a table and field in the backend database. In this case, the assistant uses the heuristics obtained for the given fields and applies a set of customizable rules used to decide which slots can be unified and which ones cannot. Some examples of the rules applied to not propose the unification are: a) one of the slots is defined as a string with an average length greater than 20 characters, an average number of words per record greater than 3, and the other slot is an integer/float number greater than 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 account numbers, b) when two slots are defined as strings and the sum of the average length of both is greater than 20 characters; in this case, the system tries to avoid the recognition of very long sentences. 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 these cases, the system decides that it is better to ask one slot at a time (system initiative). In case there are more than two slots, the system checks different combinations of the slots in order to find those that can be requested at the same time and leaving the other one to be requested alone.

6.2 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 and the database access functions, the wizard allows the designer to access to the following state proposals:

Empty states and already created states: The first one allows the creation of a new empty state, with no defined slots inside, that the designer can define completely afterwards. This way, we allow a top-down design. The second one allows the designer to re-use already defined states.

From attributes with database dependency: This kind of states is created from any attribute defined in the database model (DMA) that refers to a database field only if the attribute has been used as an input argument of any database access function. The proposed states contain only one slot and its name corresponds to the name of the attribute in the data model. However, the designer can select several states to create states with multiple slots.

From the database access functions: In this case, the system analyzes all the defined database functions 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. The assistant allows the designer to select several of these proposals in order to create more complex states. For instance, in case there is a database access function called *convertCurrencies*, which receives three input arguments (i.e. *fromCurrency*, *toCurrency*, and *Amount*), the system automatically creates a new state proposal called *convertCurrencies* that includes these three slots. Applying similar rules to the ones described in section 6.1 the system would propose to request the first two at the same time (mixed-initiative) and the *Amount* separately (directed forms).

From classes defined in the data model structure: In this case, the assistant creates a template that the designer can drag and drop into the workspace (see Figure 6 in Appendix A). Then, a pop-up window allows the designer to select the attributes to be used as slots. The assistant expands complex attributes (with inheritance and objects) allowing only the selection of atomic attributes.

7 Evaluation

With the objective of evaluating the performance of each of the acceleration strategies and assistants 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. They were requested to fulfil different typical tasks covering each of the proposed accelerations and assistants to evaluate. Further details can be obtained in (D'Haro, 2009).

For the subjective evaluation, the participants were asked to answer a questionnaire that consists of four questions per assistant and seventeen for the overall platform, with a range between 1 and 10. This subjective evaluation confirms the designer-friendliness of the assistants, as well as their usability, since all the assistants obtained a global score higher than 8.0, which is a nice result. In detail, the DMA and DCMA obtained an 8.3, the SFMA a 9.0, the RMA an 8.6, and Diagen a 4.5. Regarding the acceleration strategies, see Figure 3a, the evaluators scored the automatic states with 9.3, the SQL generation and the unification of slots for mixed initiative with 9.0, and the class proposals with 8.9. Regarding the RMA and the accelerations related with the information extracted from the database (see section 3), the passing of arguments between actions and the proposal of dialogue actions obtained a 9.8 and 8.6 respectively.

For the objective evaluation, we collected the metrics proposed in (Jung et al, 2008): elapsed time, number of clicks, number of keystrokes, and number of corrections using the keyboard (keystroke errors). We compared our assistants with a built-in editor called Diagen, created during the GEMINI project and improved later on by (Hamerich, 2008), 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 confirm that the design time can be reduced, in average for all the assistants and tasks, in more than 45%, the number of keystrokes in 81%, and the number of clicks in 40%. Especially relevant is the high reduction (85%) obtained in the RMA considering that it is the main task in the design.

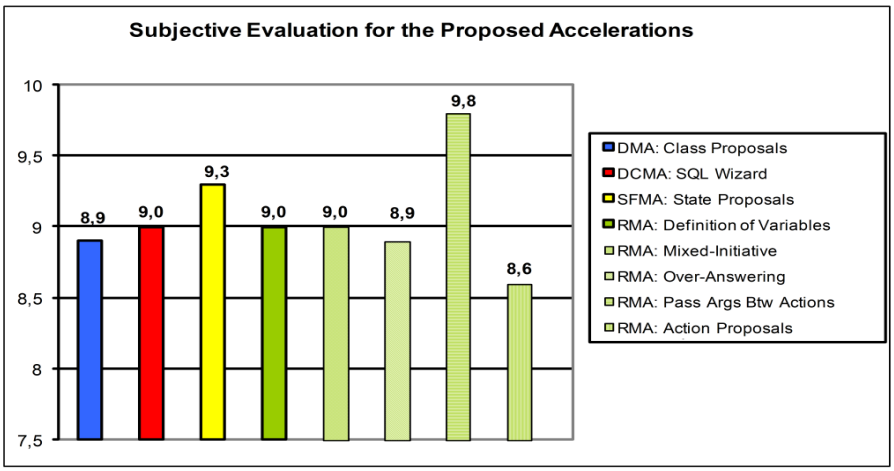
8 Conclusions and Future Work

In this paper, we have described the main accelerations incorporated into a complete platform for designing multimodal and multilingual dialogue applications. The proposed accelerations strategies are based on using information extracted from the contents of the backend database. The proposed accelerations 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 for accessing the database, among others. Subjective and objective evaluations confirm that the proposed strategies are useful and contribute to simplify and accelerate the design.

As future work, we propose the extraction of new heuristic information, the creation of new rules for unifying slots for mixed-initiative dialogues. Considering the negative values in Figure 3b, we propose to improve the GUI for defining the connections among states in the SFMA, and to improve the DCMA by offering new automated methods for creating the prototypes.

9 Acknowledgements

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



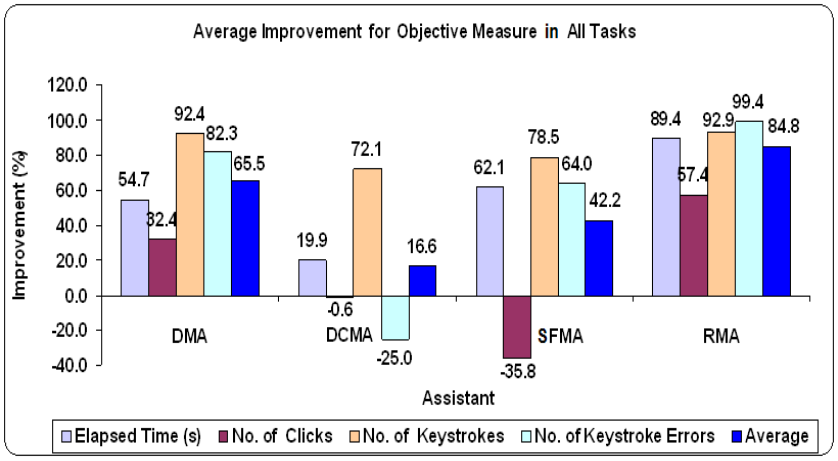


Figure 3. Average result for the: a) subjective evaluation for the accelerations, b) objective results

References

Chung, G. 2004. *Developing a Flexible Spoken Dialog System Using Simulation*. ACL 2004.

D'Haro. L. F. 2009. *Speed Up Strategies for the Creation of Multimodal and Multilingual Dialogue Systems*. PhD Thesis. Univ. Politécnica de Madrid.

D'Haro, L. F., Cordoba, R., et al. 2008. *Language Model Adaptation for a Speech to Sign Language Translation System Using Web Frequencies and a MAP framework*. Interspeech 2008, pp. 2119-2202.

D'Haro, L. F., Cordoba, R., et al. 2006. *An advanced platform to speed up the design of multilingual dialogue applications for multiple modalities* Speech Communication Vol. 48, Issue 8, pp. 863-887.

D'Haro, L. F., Cordoba, R., et al. 2004. *Strategies to reduce design time in multimodal/multilingual dialog applications*. ICSLP 2004, pp IV-3057–3060.

Feng, J., Bangalore, S., Rahim, M. 2003. *WEBTALK: Mining Websites for Automatically Building Dialog Systems.* ASRU 2003, pp. 168-173.

Hamerich, S. 2008. *From GEMINI to DiaGen: Improving Development of Speech Dialogues for Embedded Systems*. 9th SIGDIAL, pp. 92-95.

Jung, S., Lee, C., et. al. 2008. *DialogStudio : A Workbench for Data-driven Spoken Dialogue System Development and Management*. Speech Communications, 50 (8-9), pp. 683-697.

Pargellis, A. N., Kuo, H. J., Lee, C. 2004. *An automatic dialogue generation platform for personalized dialogue applications*. Speech Communication Vol. 42, pp. 329-351.

Polifroni, J. and Walker, M. 2006. *Learning Database Content for Spoken Dialogue System Design*. LREC 2006, pp. 143-148.

San-Segundo et al. 2008. *Speech to sign language translation system for Spanish*. Speech Communication Vol. 50, pp.1009–1020.

Appendix A. Additional Figures

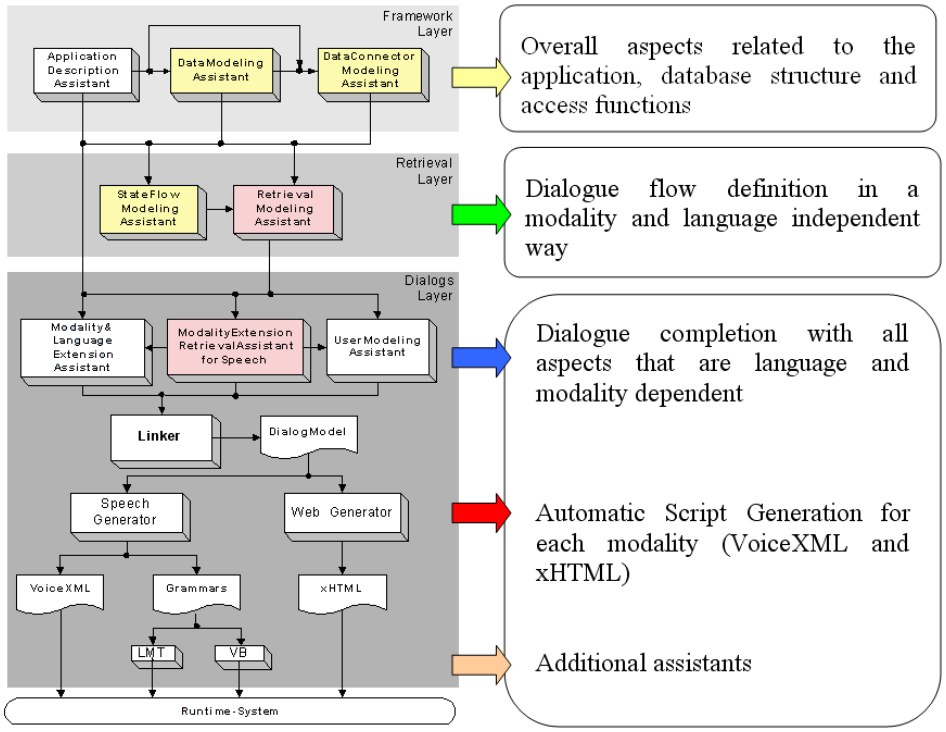


Figure 4. Platform architecture. In yellow colour the assistants with the new accelerations described in this paper. In pink colour assistants with previous accelerations (section 3)

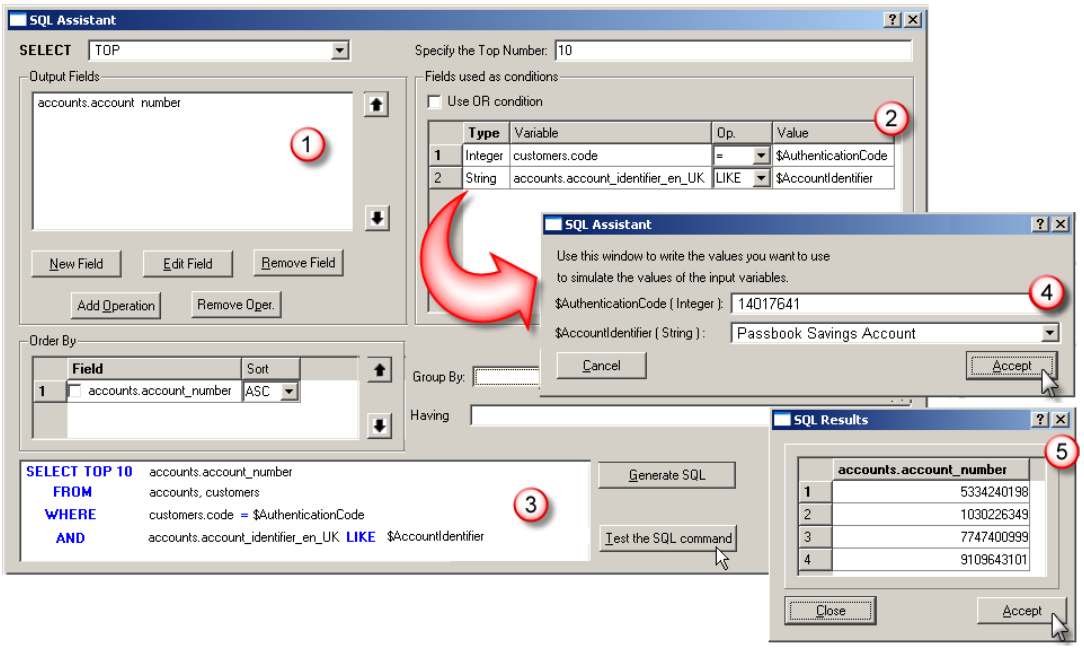


Figure 5. Wizard for creating and debugging the SQL statements for accessing the backend database. In the example, the proposed query allows the selection of all account numbers for a given customer (using his/her authentication code) and type of account (i.e. passbook saving accounts)

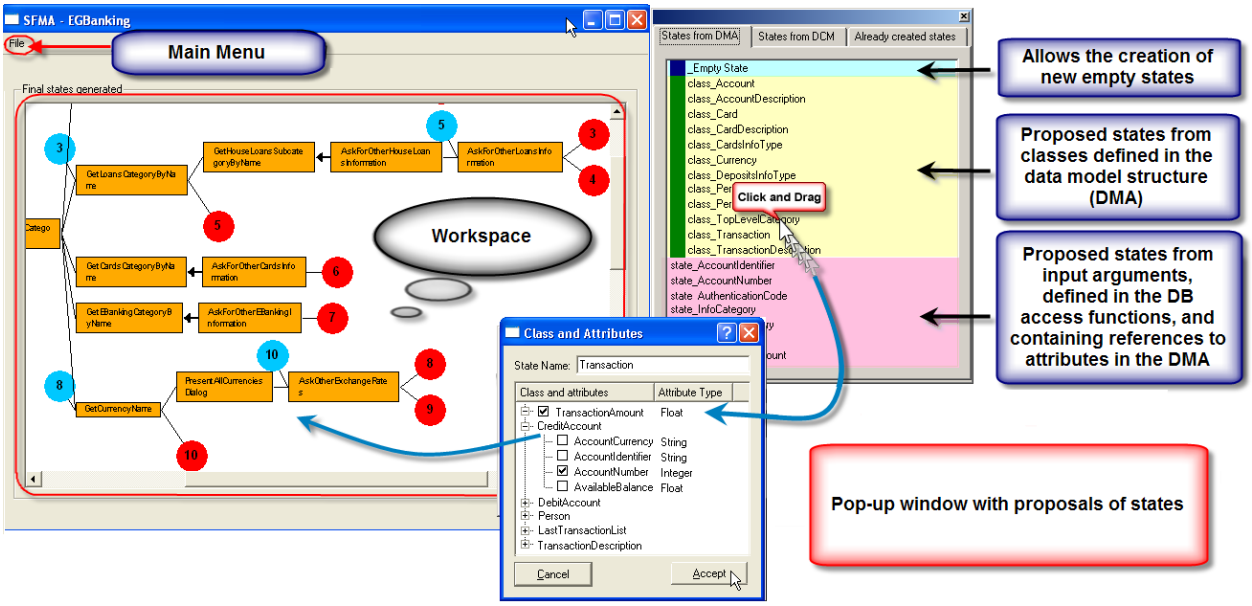


Figure 6. Workspace for creating the state transition network and pop up window with state proposals. In the example, the designer creates the state *Transaction* from the *Class_Transaction* template (created in the DMA, see Figure 2) and selects as slots the *TransactionAmount*, *CreditAccountNumber* and *DebitAccountNumber* (not shown)