International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:6, 2007

An Enhanced Tool for Implementing Dialogue Forms in Conversational Applications

Ilias Spais and George Bafas

Abstract—Natural Language Understanding Systems (NLU) will not be widely deployed unless they are technically mature and cost effective to develop. Cost effective development hinges on the availability of tools and techniques enabling the rapid production of NLU applications through minimal human resources. Further, these tools and techniques should allow quick development of applications in a user friendly way and should be easy to upgrade in order to continuously follow the evolving technologies and standards. This paper presents a visual tool for the structuring and editing of dialog forms, the key element of driving conversation in NLU applications based on IBM technology. The main focus is given on the basic component used to describe Human – Machine interactions of that kind, the Dialogue Manager. In essence, the description of a tool that enables the visual representation of the Dialogue Manager mainly during the implementation phase is illustrated.

Keywords—Conversational Applications, Forms Dialogue Manager (FDM), Natural Language Processing, Natural Language Understanding.

I. INTRODUCTION

NATURAL language Understanding (NLU) holds a big promise towards achieving a new model of human machine interaction. NLU systems are expected to give rise to a new wave of conversational applications where simple spoken command interfaces to interactions resembling talking to a human operator.

Conversational natural language understanding is a way of interacting with a machine (e.g. computer), based on Speech Recognition and Text To Speech theory. NLU gives the user the opportunity to have a conversation with the machine, in which spoken commands are given, and in response the user hears either the answer to his query, or a request for disambiguation and clarification. The machine understands the task the user tries to accomplish, and uses a dialog to help him reach his goal quickly and accurately.

Based on NLU approach, several conversational NLU applications (e.g. telephony applications) can be or already have been, implemented. The overall objective of such an application is to understand requests, translate them into actions and provide an appropriate response (in the telephony context for example, the responses are audio prompts). Besides this, it allows the user to take initiative or control over the dialog as and when desired, to speak in a natural way and to let the dialog flow in a prescribed manner. The implementation is not rule based unlike the directed dialog systems. It is statistics based and is data driven.

Taking into account the above hints, we can assume that NLU systems could potentially allow the development of conversational applications for complex and specialized tasks that today require specialized operator personnel.

The latter direction hinges on developing new tools, techniques and frameworks supporting conversational application development. This paper describes the design of a visual tool (called NLUForms GUI) enabling rapid development of form documents driving NLU dialogs. This particular design is tailored to IBM's NLU systems, and takes into account their operation and specification. Having as basic feature its authoring and managing capabilities this tool can be used by NLU systems in order to implement and deploy applications in various applications domains (e.g., finance, travel, information access (portals)) and multiple languages (e.g., English, German, Spanish, Greek, Finish).

The major motivation behind providing this design was to alleviate the complexity of NLU application development. A crucial component of the NLU architecture is the Forms Dialog Manager, which coordinates dialog interactions. The operation of this component is based on FDM files that are represented in a particular file format. Authoring and managing such files is a tedious task for NLU developers. The authoring part described in this paper aims to the development of FDM files and the target of the managing part is to import FDM files that are already have been developed. These two features of the tool provide the developer the power to create FDM files from scratch and combine them with existing ones in order to develop complemented applications. Having this kind of a tool that eases the structuring and development of FDM forms is certainly a contribution to the faster development of conversational NLU applications for various domains, with lesser human resources and as a result with lower cost.

The paper has the following structure: Section 2 presents the architecture of IBM's NLU systems. This architecture is presented not only for completeness reasons but mainly to facilitate the understanding of the design concepts presented in this contribution. Section 3 concentrates on basic design and functionality concepts of this visual tool, while section 4 deals with the implementation of this environment based on a WinIntel platform. Finally, Section 5 concludes the paper and lists ideas for extending this work.

II. ARCHITECTURE OF THE CONVERSATIONAL NLU SYSTEM

A. System overview

Fig.1 gives the overall picture of a natural language understanding system, operating in the scope of a telephony application [9].

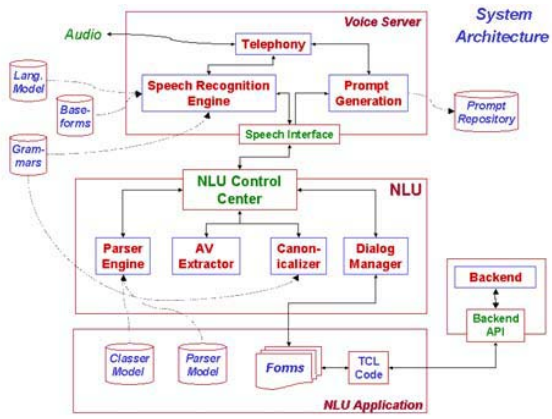

Fig.1. System Architecture

The user of the system submits his/her requests by using the telephone network. In order to perform its objectives the system consists of the following major components: Voice Server, NLU System and NLU Application. The *Voice Server* component performs all tasks related to *Telephony* - the interface with the telephone network, *Speech Recognition* - the translation of spoken requests into text, and *Prompt Generation* – the process of converting textual prompts into audio. The *NLU System* performs the task of understanding user requests, extracting their meaning, translating them into actions (or action requests) and providing an appropriate response. It does this in conjunction with application code/data and backend logic. The NLU System consists of the following run-time components: NLU Control Center, Parser Engine, Attribute-Value (AV) Extractor, Canonicalizer and Dialog Manager. The *NLU Application* component of the Conversational System consists of the Classer and Parser Models, the Forms and the Code, which is the connection with the backend (i.e. TCL Code). In addition, the Grammars are also a part of the application. Most of the work in building the Conversational System concerns developing and fine-tuning these components.

B. NLU System - Dialog Manager

The most important component of NLU System is the Dialog Manager. The Dialog Manager or the Forms Dialog Manager (FDM) as it sometimes called, takes the actions and parameters that were deciphered from the user request by the classer, parser, Canonicalizer and AVExtractor and decides on the task to be performed. It typically performs one or more of the following three tasks:

- Return a prompt to be played to the user
- Request information from the user
- Make a request to the backend, typically to: a) Disambiguate a user specified value, b) Perform some action, c) Ask the backend for some information

Thus the Dialog Manager plays a very significant role in the overall Conversational System.

C. NLU Application - Dialog Manager's Forms

The Dialog Manager performs its tasks using a set of application specific code called *Forms*. The application specific actions that are to be performed in response to user requests are defined in the Forms. Typically there is a form file for each major transaction in the system. Forms are written using form-specific syntax (*FDM script*) and each form makes calls to functions, which can directly connect to the backend and execute user's request.

Forms consist of "Slots" and "Messages". Slots are fields for parameters or classes that define the action to be performed, while Messages are function-statements that are called to perform one of the three Dialog Manager tasks listed above. In order to determine which form gets the focus, the FDM takes as input attribute-value pairs from the parser, fills the slots, calls the application to see if the slot values are valid, and applies extensive rules to help it determine which form to set the focus to. In order to "submit" the form to the back end, the FDM works on filling slots until it can effectively submit the information (i.e. all mandatory information elements have been supplied). When the FDM has enough slots filled, the FDM "submits" to the back end by firing an event called a backend message "BEMsg". Firing this event causes backend business logic to execute. As an example, let's consider a user's request 'What is my balance in blue chip growth' (Fig. 2). The basic form would have a slot for the fund-name and several messages: a) a message for disambiguating the fund-name, b) a message seeking clarification/information from the user and c) a message to get the actual balance in the fund from the backend and return it to the user [9]. When there is more than one suitable form, Dialog Manager decides which form to use (as basic form), taking into account how many slots each form has filled with the appropriate parameters.

Finally, there are 2 main types of messages: a) The *backend* message, which executes logic and determines which form or slot to set focus to and which message to fire next and b) The *prompt* message, which causes the FDM to wait for user input (wait for new attribute-value pairs from the parser).

D. Form's Specific Syntax (FDM Script)

FDM scripts are flexibly designed to include the information elements required to implement the fully fledged functionality of the Dialog Manager. Specifically, FDM scripts consist of three basic components (Fig. 3). First and foremost, there is the *begin{form}* line which defines the first form included in this file (there may be more than one form) and the name of the form. Besides this, this block specifies the

relationships (as a result of inheritance) that this form has. The end of the form is determined by the *end{form}*. Secondly, there is the slot block which contains the slots and their properties. The lines of the block are determined by the keywords *\begin{slots}* and *\end{slots}*, which are its boundaries. The basic properties of this block are the *Matchedby* and the *Inherit*. The first one determines when the specific slot will be fulfilled and the second one shows its relationships with the other slots.

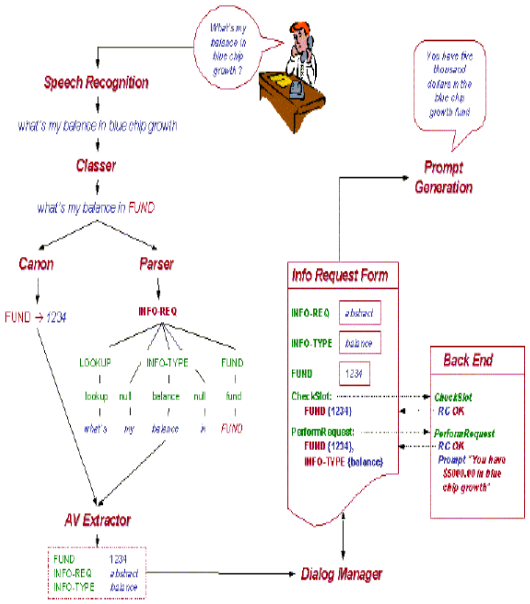

Fig.2. Schematic of NLU Runtime Processing

Note that users often supply ambiguous information. This could happen because either they do not know how "*truth*" is represented in the database (application) or because they do not know what choices they have. Clarifying the ambiguity highly depends on what is in the database, as well as on the user profile – both of which can change rapidly. The final authority on what is ambiguous and what is not is the application itself. Calls for backend consultation are happening when filling some slots (*called abstract slots*). A slot requires backend consultation if the forms file has a slot-level backend message in the description of the slot (Fig. 3). Note however that some slots - such as dates, times, money amounts - can be filled without backend consultation (*concrete slots*). Furthermore, the slot-level disambiguation message is the only message to the backend at the slot-level. It is really no different from any backend message in terms of protocol and of how the FDM reacts to the return codes from the backend.

Last but not least, is the message block (Form Level Messages) which contains form level messages and their properties. The boundaries of the messages are determined by the lines *\begin{messages}* and *\end{messages}*. The basic properties of the block are shown in the Fig. 3 (words in bold).

As soon as enough slots have been filled out, the form reaches status *'OK'* and constructs the backend message. This message consists of calls to functions (actually written in the TCL code). These functions retrieve the replies from the backend along with a list of possible answers for the user depending on the answer from the backend. The list is being confirmed by the keywords begin{rclist} and end{rclist}. Supposing that a non empty set of results is retrieved the system presents the answer to the user and waits for the next request.

```
\begin{form} FORM-NAME     ^Inherit: ……..
#---------------------------------Slots-----------------------------------
-----------------
     \begin{slots}
       \slot  SLOT-NAME MatchedBy:…….
       \slot  SLOT-NAME  ^MatchedBy:……
                         ^Inherit:……
              \begin{messages}
                   \msg Prompt:……;
                   \msg BEMsg:
              \end{messages}
     \end{slots}
#-------------------------------------------------------------------------
#----------------------- Form Level Messages---------------------
       \begin{messages}
         \msg BEMsg: required="……."
            #    (id,required,inhibited,freshness)
            TCL-code  string $Slot-name
            \begin{rclist}
               \rc…… OK……...\msg Prompt:……;
               \end{rclist}
          \msg HelpMsg:…….
       \end{messages}
#-------------------------------------------------------------------------
 \end{form}
```

Fig.3. FDM File Format

E. Backend

The backend refers to the system's module where the actual transactions and information requests are carried out. The backend could be as simple as a database or it could be a complex system with an interface for the Conversational front end. The backend system is usually outside of the Conversational system.

III. DESIGNING NLUFORMS GUI

A. Functional Overview

Based on the NLU/FDM architecture and operation, the NLUForms tool enables visual development of FDM scripts for use by Dialog Manager of Natural Language Understanding NLU systems. The framework of this tool allows editing of FDM data components, through an appropriate Graphical User Interface (GUI). Furthermore, it supports visual creation of FDM scripts (from scratch), while providing commands for editing and processing FDM scripts. In addition to, it saves an application in the form of a *workspace,* so that application developers can work on it at any time instant. Besides this, the environment allows the

developers to import and work with forms that were not originally created by this tool, provided however that they comply with the FDM specification. Thus, users can create sets of NUL application forms either by importing existing forms, or even by creating new forms. It is also feasible that the import and create capabilities of the environment are combined. Finally, NLUForms GUI incorporates a host of utilities easing the manipulation of FDM data elements and components. This is accomplished through an appropriate set of menu commands, dialog boxes and screens used for inputting data. Even though the environment provides a user friendly interface, potential users must have a basic understanding of the NLU/FDM architecture, component and component types.

B. Workspaces Support - Workspace Configuration

The notion of a complete authoring environment, as an NLU developer, is expressed through support for workspaces. Within a workspace the user/author can edit multiple NLU Form objects. The rationale for providing such functionality becomes apparent since:

1. The vast majority of NLU applications consist of more than one Form and corresponding FDM scripts.
2. The forms of an NLU application present relationships (e.g., inheritance) affecting FDM generation.
3. FDM authors may need to share information pertaining to all the forms of the NLU applications. This common information relates both to information entities (e.g., Classer Parser keywords, Functions available at the message template editor facility) and editing options (e.g., FDM generation preferences).

Each instance of a particular workspace is associated with several options governing the process of Form's authoring and FDM generation. Characteristic examples of such options include the number of Form objects that can be edited in the scope of the workspace, the directory where FDM scripts should be placed during batch generation of FDM scripts and the path to an executable file corresponding to an editor/viewer for FDM text [4].

Thus, the process of saving the authoring work concerning a particular NLU application, involves storing the above options as well. Specifically, when a user elects to save the workspace, Form objects and their components (i.e., Slot & Message objects and their encapsulated information entities) are serialized in a file along with the workspace options. Therefore, the file that stores all this information is characterized as a *workspace file*.

C. Import Functionality

Import is one of the basic functionalities that NLUForms GUI can offer to any NLU developer. This ability comprises commands for importing existing FDM forms that were not generated by the environment and consequently do not belong to any workspace. Thus, the author can also use the GUI to create NLU applications by only importing forms. He can also create forms on his own and then fill in his application by using import functionality. The GUI has also the ability to import more than one form, which may be included in the same physical file.

Forms must be at a specific format as it was described above. The form must contain specific keywords (bold words – Fig. 3). Keywords are used by the program, in order to define the name of the form, the names of the slots, the names of the messages, the inherited forms and all the properties of the form. Furthermore, the form must have the slot level and the message level blocks of the file separated explicitly. As the import functionality is being carried out, the program is reading the file (form file – FDM script) and finds out the position of the keywords. In this way it manages to figure out the basic pattern of the form and then it defines the properties of the form. As someone can easily imagine, knowing the file format is something extremely important at this specific stage. Finally, the imported form is being added at the end of the application which is being transformed by the user of the GUI. This last form has the focus just after the import functionality has succeeded, in order to give the user the opportunity to check its Slot and Message Objects.

D. Graphical User Interface Design

As far as the user interface design is concerned, it is centred on a conventional (main) application window, which is split into four different panes (Fig. 4). Each of the four panes constitutes a different display area that serves a distinct purpose.

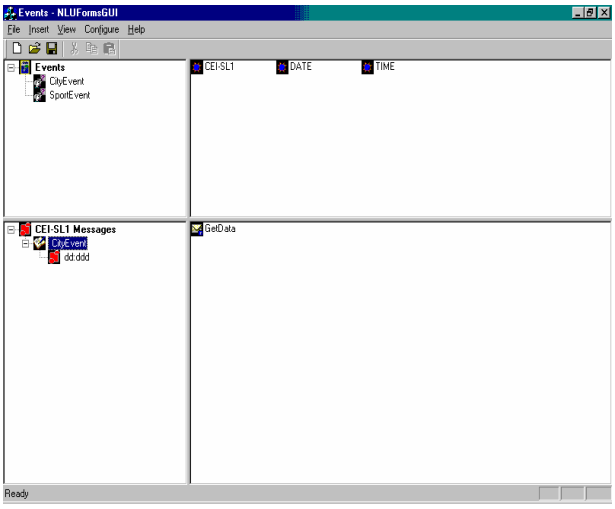

Fig.4. Overall view of the main application window

Specifically:

- The top left pane, displays a set of Form Objects (corresponding to FDM scripts being edited), based on a Tree layout. On this layout one script (denoting the active one) is highlighted. The tree layout is used towards depicting form inheritance.
- The top right pane is devoted to displaying Slot objects

(i.e. NLU slots) belonging to the active (if any) FDM script. Slots are designed to be displayed as a list.

- The bottom right pane acts as a display panel for Form Level Message Objects. Like in the case of the slots pane Form level messages are displayed as a list.
- The bottom left pane depicts Message objects (i.e. message templates) belonging either to the active (if any) slots (disambiguation/confirmation messages) or to the active form level message (function statements and function calls). Basically, this pane displays messages based on which other pane is active at the moment. In this pane messages are displayed based on a Tree layout.

The environment's commands are designed to be available through menus. There are two types of menus: these accessible from the menu bar, and context sensitive menus.

Finally, the environment comprises a rich set of Dialog Boxes. These Dialog Boxes are user interface elements enabling data input, editing of object properties, user interaction. Dialog boxes pop up as a result of menu commands, or commands issued by pressing/clicking buttons on other dialog boxes.

IV. IMPLEMENTATION

So far we have described the basic theoretical parts of NLUForms GUI, what it can do and how. Following the functional description and basic design structure of the authoring environment, we provide some details about its software implementation. This implementation was carried out using MS Visual C++ and relying in the MFC libraries. SDI (Single Document Interface) architecture was selected for implementation. Single Document Interface is a document – centric application, at the heart of the MFC framework that can only work with one document at a time, and can only work with one type of document [7].

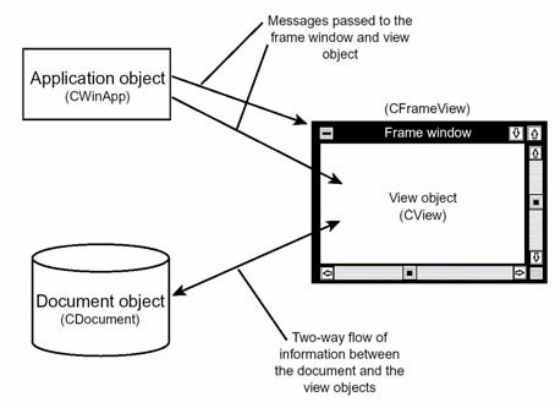

Fig.5. The SDI architecture.

There are four classes that make an SDI application work (Fig.5):

- *CWinApp-derived class*. This class: a) creates all the components of the application, b) receives all the event messages and c) passes the messages to the *CFrameView* and *CView* classes.
- *CFrameView-derived class*. This class is the window frame. It holds the menu, toolbar, scrollbars, and any other visible object which is attached to the frame. This class determines the visible part of the document at any time.
- *CDocument-derived class*. This class houses a document and manipulates the data that makes it up. It also receives input from the *CView* class and passes display information to the *CView* class. Finally, it is responsible for saving and retrieving the document data from files.
- *CView-derived class*. This class displays the visual representation of the document for the user. It also passes input information to the *CDocument* class and receives display information from the *CDocument* class.

As far as our project is concerned *CNLUFormsGUIDoc* is the class that acts like the *CDocument* class that we described above. To be more specific, this class processes the data of each form at a time. Furthermore, it calls *CNLUForm*, *CNLUSlot* and *CNLUMsg* classes. These classes manipulate for each application forms, slots and messages respectively.

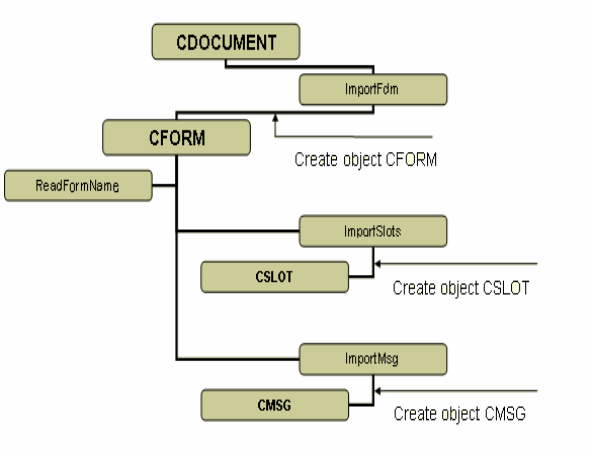

Fig.6. Import Functionality.

All the classes with the extension *View* extend the *CView* class. For example *CFormTreeView* displays the tree-like visual representation of the active form and *CSlotListView* displays the list-like visual representation of slots.

Finally *CMainFrame* is responsible for creating the four panes of the main application window. As an example of how classes cooperate, Fig. 6 presents the implementation of import functionality based on SDI architecture.

V. CONCLUSIONS – FUTURE WORK

In this paper we have presented significant design and implementation aspects of a visual tool supporting the development of NLU forms. The framework of this tool supports the notion of a Forms workspace. In the scope of such a workspace the NLU developer can create and edit

multiple FDM scripts. NLUForms GUI allows users to develop FDM forms from scratch, but also to import and edit (pre)existing FDM scripts that may have been manually developed. It is also important that the environment provides a host of other operations that facilitate the development of FDM scripts. Note that all operations are visual and supported by a user-friendly graphical user interface.

As the Dialog Manager is among the most important components of the Conversational Natural Language Understanding System, the benefits from using this environment are straightforward. In particular, using this environment NLU developers are alleviated from the tedious task of manually writing FDM scripts. Manual authoring is slower, less intuitive, more error-prone and overall less productive.

It is beyond any doubt that FDM is *form-based* because each task that can be performed is associated with a form. To perform the above, the FDM decides which form any particular user utterance is referring to. Relevant information needed to specify the task is the data fields, which we refer to as slots of that form. One may think of FDM as aiding the user in picking out a form and filling the appropriate slots. The actual execution of tasks is performed by a separate component of the system called the backend.

The presented effort is very closely related to a number of other tools and frameworks supporting NLU development. It is a subject of future work to revise the implementation so that this GUI could become part of a wider NLU development toolkit.

REFERENCES

[1] M. Epstein, 'Statistical Source Channel Models for Natural Language Understanding', Phd Thesis, NYU University, Department of Computer Science, 1996.

[2] T. Ward, S.Roukos, C. Neti et. al. 'Towards Speech Understanding across Multiple Languages' in the Proc. of the 5th International Language Processing, ICSLP 1998, Sydney Australia.

[3] K. A. Papineni, S. Roukos, and R. T. Ward, 'Free-Flow Dialog Management Using Forms'. Eurospeech 99, Budapest, Hungary, 1999.

[4] John Soldatos, Valia Demesticha 'Visual FDM Builder Version 1.0 User's Manual', May 2002

[5] Marion Mast, Thomas Ross, Henrik Schulz, Heli Harrikari, Vasiliki Demesticha, Lazaros Polymenakos, Yannis Vamvakoulas, Jan Stadermann: 'A Conversational Natural Language Understanding Information System for Multiple Languages' in the Proc. of the 6th International Workshop on Applications of Natural Language for Information Systems Conference, Madrin Spain, June28-29, 2001, pp.177-186

[6] CATCH-2004 "Converse in Athens, Cologne and Helsinki", Annex I "Description of Work", IST Programme - Key Action III, Proposal number IST-1999-11103.

[7] Jeff Prosise, 'Programming Windows with MFC', Microsoft Press 2nd Edition, May, 1999, ISBN: 1572316950.

[8] VoiceXML 2.0, W3C, http://www.w3.org/TR/2001/WD-voicexml20-20011023, Working Draft.

[9] IBM, ViaVoice, http://www-3.ibm.com/software/speech/