Intuitive Dialogue Flows Design for Conversational Interfaces

Stefano Valtolina
Department of Computer Science
Università degli Studi di Milano
Via Celoria, 18 Milano, Italy
valtolin@di.unimi.it

Lorenzo Neri
Department of Computer Science
Università degli Studi di Milano
Via Celoria, 18 Milano, Italy
lorenzo.neri@studenti.unimi.it

ABSTRACT

Nowadays, new technologies are taking place for enabling conversational interactions between users and bots. Experts in mobility, environment, energy, culture, e-health, weather, etc., are the only ones who well-know the specific domain in which the bot will act. However, the design and implementation of the interactive flow of dialogue cannot be properly defined by them because deep programming skills are needed. Starting from this consideration, the paper proposes a preliminary system that offers to domain experts the possibility to design a flow of dialogue without getting lost in technicalities. Using a visual language, the domain expert can specify intents, actions, entities and parameters at the base of the flow of dialogue on which the bot will be created. Then, our engine automatically translates in a bot complaint with the DialogFlow technology.

CHI'19 Extended Abstracts, May 4-9, 2019, Glasgow, Scotland, UK.
© 2019 Copyright is held by the author/owner(s).
ACM ISBN 978-1-4503-5971-9/19/05.
DOI: https://doi.org/10.1145/3290607.XXXXXXX

KEYWORDS
Chatbots; Conversational interfaces; Design trade-off; User inclusion; Design strategy.

INTRODUCTION

User-Centred Design (UCD) is a design strategy that aims at satisfying end-users' needs and desires through a usable interface and an effective navigational experience. UCD strategy relies on a combination of research and user experience (UX) design activities. In this case, UX deals with the specific experience users have with the products they have to use.

The design of a web-based interactive application can put the user at the centre of the project, not only improves her/his experience but also reduces costs. A successful system avoids any future interventions addressed to meet needs that can arise after its actual use. In an attempt to make the human-computer interaction more efficient, designers continuously try different approaches as in the case of tools like Amazon Echo and Alexa, Google Home and Siri, where new interaction strategies are proposed based on the dialogue between users and applications by simply using their voice.

Even if UCD has always focused on designing usable GUIs (Graphical User Interfaces) and UX has been closely related to the world of visual perception, the design of conversational interfaces requires to focus the attention on the dialogue and to find in the linguistics, pragmatics and semantics the scope of action as the graphical interfaces find it in the world of vision and semiotics [1]. Designing a conversational interface without understanding how humans converse and talk, it is equivalent to build a visual user interface without any basis of visual perception.

From a technical point of view, current conversational interfaces allow users to speak or to chat with bots using voice (voicebot) or text (chatbot) [2] [3].

These conversational interfaces lack or provide few visual elements (e.g. buttons, slides, pop-down menus) and we have to rely on the dialogue to express states and processes or to navigate the system. For this reason, the interaction is usually a linear flow or a flow with few ramifications. Today several domains can benefit from the use of voice or chatbots: e-commerce, health-care assistants, customer/citizen service systems, or IoT device management. In these contexts, bots can provide a direct and simple entry point for the user requiring information or suggestions for selecting a product or activating a device. The flow of dialogue is designed with applications called conversational design editors with which designers specify the way the bot has to react. The bot engine exploits this flow of the dialogue and the related retrieving strategies for accessing information or for activating devices.

The design of such flows is typically done by ICT experts who have the required technical skills to develop the dialogue and the related algorithm around which the bot will be developed. By engaging domain experts, the quality of the dialogue will be improved because they have the necessary experience and knowledge to highlight real and effective dialogues. However, at the current stage, they need to be supported by ICT experts and together they have to develop the flows.

This paper aims at presenting a preliminary study about the design of a conversation interface editor that supports non-technical domain experts in creating a dialogue flow and automatically generate the related bot without the support of ICT experts.

In details, suppose that experts in public transportation and urban mobility need to create a bot to help citizen to reach a specific location in the city. The bot leads the users in conversion asking her/him to choose different transportation strategies. For example, it could suggest taking the bus but getting down at a specific stop to walk a bit, or it could suggest parking the car in a specific place and to use public transportation instead of the private one. Recommendations that the bot has to provide by taking into account the weather condition or eventually the concomitant presence of major events in the city. The dialogue to design is not a simple direct flow of conversation but need to take into account the user's answer and the information the bot needs to retrieve from external APIs. APIs to use to understand the weather conditions or the presence of major events in the city on the given date.

In the first section, we describe how our contribution stems from an End-User Development (EUD) definition. We present the EUD as a design method able to provide domain experts with unwittingly developing strategies to generate personalized flows of dialogues. The second section describes the conversational interface editor and the technical details that needed to develop a bot on *Dialogflow*. Section 3 presents the visual language used to support domain experts to create flows and to automatically translate them in real bots without writing any line of code. Finally, the last section tracks some conclusions and future works.

EUD AND CONVERSATIONAL INTERFACES (CI)

The definition of end-user interfaces has experienced deep changes in the last decade. However, some seminal works in the consolidated EUD scientific literature still hold and are those that see the end-user as someone interested in using digital devices just for the sake of it and not with the idea of becoming expert in the technology itself (e.g. [4], [5]). Also, the definition of EUD given in [6] still sounds valid to describe the phenomenon: "*a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact*".

From an organizational point of view, end users are not necessarily experts in computer science, but in the domains, they work in. Nardi's definition given in [7] states that the end-user is "*the person who does not want to turn a task into a programming problem, who would rather follow a lengthy but well-known set of procedures to get the job done*". With the large diffusion of virtual agents, conversation interfaces are becoming one of the most adopted interaction paradigms in which information and services are seamlessly available by using a natural interaction such as the dialogue. Brancheau and Brown [8] confined the end-users to a space that is somewhere "*outside the information system department*". The idea is to confine end-users at the end of the design and the development processes and to put distance, as suggested by Cypher [9], between them.

This approach does not reflect the current requirements that the design of conversational interfaces need to deal with. Conversational interface design tools should provide end-users with resources that support them in the creation of a bot. In detail, end-user are the domain experts (experts in mobility, environment, energy, culture, e-health, weather, etc.,) who have the expertise to design an effective dialogue because they well-know the specific domain in which the bot will act.

CI EDITOR FOR DOMAIN EXPERTS

We analyzed the most diffused applications for supporting users in designing voice or chatbots that include: Botsify (*botsify.com*), Chatfuel (*chatfuel.com*), Chatbot.com (*www.chatbot.com*), ChatScript (*github.com/ChatScript/*), IBM Watson Conversation (*www.ibm.com/cloud/watson-assistant*), ManyChat (*manychat.com*), Rasa (*rasa.com*), Recast.ai (*cai.tools.sap/*), Wit.ai (*wit.ai/*).

All these solutions suffer from the same common limitations. In some cases, they allow the user to design single flows with no possibility to create ramifications in the dialogue. Other solutions, like *Chatfuel*, do not offer the possibility to query external APIs that can be used by the bot to provide further information required to complete the conversation (about the weather, the availability of parking, public transportation, information about a product or service...). Tools like *Chatfuel*, *Chatbot.com*, *Wit.ai* offer the possibility to create bots for only specific platforms such as *Facebook* or *Messenger*. However, several of these solutions require the designer to know a specific programming language and technicalities for creating effective and intelligent bots. To overcome these problems, our proposal aims at providing domain experts with a graphical editor for the design of a bot by using visual elements and for translating it automatically in an effective bot. The design environment is based on *Draw.io (app.diagrams.net/).*

Draw.io is a web-based open-source technology that allows you to build diagrams in a very simple matter. The final bot is implemented in *DialogFlow (cloud.google.com/dialogflow/docs)*. *DialogFlow* is a natural language processing (NLP) platform provided by Google corporation that can be used to build conversational applications and experiences on multiple platforms (e.g. *FaceBook*, *Messenger*) or devices (e.g. *Google Home*). Developers are supported by tools to enhance their app's interaction features through AI-powered text and voice discussions. By using *Dialogflow*, developers can focus on other relevant parts of the creative process while the platform handles the standard protocols and functionalities to implement the bot. *Dialogflow* features an in-line editor so that developers can write the code directly from the console. In this platform, a developer has to specify a set of intents that are used to define how to map user's input to a corresponding bot's response. For each bot, the developer defines many intents that when combined generate a complete conversation. A basic intent contains the components reported in Table 1. As depicted in Fig. 1, every intent has a built-in response handler that can return responses after the intent is matched.

Table 1: Basic components in *DialoFlow for creating a bot*

Training phrases	Samples of the phrases that the users can say,
Action and Parameter s	Training phrases can be annotated with entities or categories of data that the developer wishes *Dialogflow* to match. They have the purpose of improving the intent's language model.
Responses	Samples of the text, speech, or visual responses to provide to the users, which usually prompts users in a way that lets them know what to say next or that the conversation is ending up,

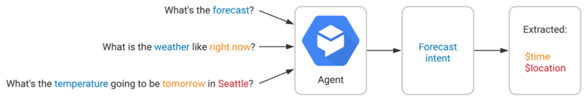

What's the forecast?
What is the weather like right now?
What's the temperature going to be tomorrow in Seattle?
Agent
Forecast intent
Extracted:
$time
$location

Figure 1: Information to specify for creating an intent. For each agent, she/he to define many training phrases by using which the bot will create the final answer (from DialogFlow - cloud.google.com/dialogflow/docs).

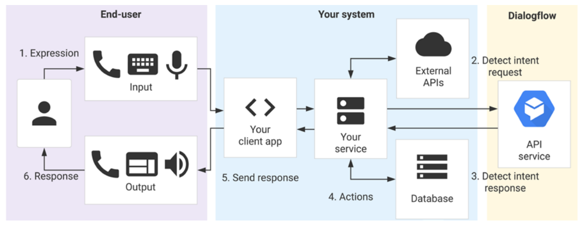


Figure2: The diagram shows the processing flow when the bot interacts with the API. (from DialogFlow documentation).

When the developer implements the intent, she/he has to establish which parameters to use for extracting useful information from the end-user expression, like a time or location for the desired weather forecast. This extracted data is important for the bot to perform a query for the end-user. When the developer wishes to create an external interaction (e.g. access a database or external API) she/he needs to create a *fulfillment* to process the intent first, and then returns a more intelligent or useful response. A *fulfillment* is a custom logic written in *JavaScript* that the developer implements as a webhook to use for handling an external interaction, processing them, and returning responses. Moreover, the developer has to specify the contexts. A context represents the current state of a user's request and allows the bot to carry information from one intent to another. The developer can use combinations of input and output contexts to control the conversational path the user takes through the dialogue. Finally, events allow the developer to invoke intents based on something that has happened instead of what a user communicates.

Dialogflow supports events from several platforms (like *Google Assistant, Slack*, and more) based on actions users take on those platforms. Moreover, a developer can also create her/his custom events that can be triggered via *fulfillment* or the proper API. As depicted in Fig. 2, the developer can design a conversational flow that directly interacts with *Dialogflow API* or external API/database to send end-user expressions and receive intent matches.

DESIGNING CONVERSATIONAL FLOWS IN DRAW.IO

Draw.io offers widgets to create graphs such as the flowchart depicted in Fig. 3. These widgets can be used to visualize the steps, decisions and the interactions needed to support the dialogue with the bot to implement. In our work, we have considered some *Draw.io* widgets for the graphical specification of the intents, external interaction, fulfillments, and contexts. By their composition, a graph is obtained that specifies the interaction in a specific context of use. The resulting graph can be serialized in an XML document which is the input of our engine that translates the graph in a proper bot written in *DialogFlow*. The engine is based on *NodeJS* that extracts the information from the XML graph created with *Draw.io*, transforms it in a JSON payload and at the end calls the *DialogFlow* API to create the proper bot. Once the engine has parsed the XML file, it starts to create a JSON object for each intent, then a *JavaScript* file for each invocation to external APIs. Then, the engine translates the flow of dialogue in a hierarchical JSON object following the arrows designed by the expert and finally, it calls the *Dialogflow API* to bring to life the bot.

In our strategy, in *Draw.io*, we ask domain experts to design the graph to make clear to the engine which intents, actions and external API calls to parse. For each intent, the user has to drag and drop on the *Draw.io* canvas, a rectangle, which a proper name (see Fig. 4). In the first section (Fig. 4A), she/he to specify a set of training phrases to use to start the dialogue. By using the different colours the user indicates the parameters the bot will use to understand the input phrases. The parameters are then written in section two (Fig. 4B). If the training phrases contain more the one parameter, the expert has to use different colours to highlight them and separate the paraments by using the symbol **§**. Finally, in the third section (Fig. 4C), the expert has to indicate the bot reply by using the parameters specified above.

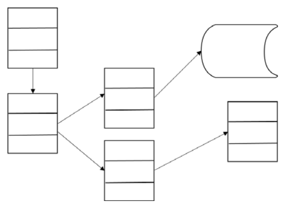
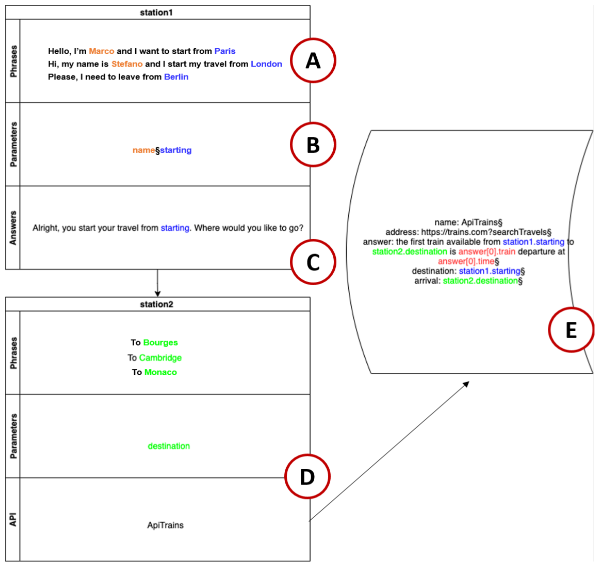


Figure 2: An example of the graph the domain expert can design to specify the flow of dialogue in *Draw.io*



Hello, I'm Marco and I want to start from Paris
Hi, my name is Stefano and I start my travel from London
Please, I need to leave from Berlin

Alright, you start your travel from starting. Where would you like to go?

To Bourges
To Cambridge
To Monaco

name: ApiTrains§
address: https://trains.com?searchTravels§
answer: the first train available from station1.starting to station2.destination is answer[0].train departure at answer[0].time§
destination: station1.starting§
arrival: station2.destination§

Figure 3: Example of a flow of dialogue in which the domain expert has created two intents. For each intent, she/he has specified the training phrases, the parameters and the bot reply. In the second intent for creating the final answer, the expert specifies to call an external API.

The actions, the contexts and the right flow of the dialogue are specified connecting the rectangles with arrows as depicted in Fig. 4. The parameters can be used not only to provide a direct answer to the user but also to invoke external APIs. In Fig. 4(D) the expert has specified as the second bot reply is created by invoking an API named ApiTrains.

By dragging and dropping a curved rectangle (Fig. 4 (E)), the expert indicates the address of the API and how the bot uses it for replying. By using the coloured parameters and the dot notation adopted for specifying the name of the section from which the parameters are established, the expert composes the final answer. In details, the bot in Fig. 4 will be activated by the final user for asking information about a trip in train. As first intent, she/he will say from which train station she/he will start. In case, she/he can also present her/himself. The bot will reply composing the answer by using the paraments: *name* and *starting*. Then, the bot will ask to indicate the station of destination. The final answer will be created by calling an external API named *ApiTrains*. By using this API, the bot will be able to indicate the first train available from the starting station to the destination as requested by the user.

CONCLUSIONS

In this paper, we have presented a preliminary system that aims at supporting domain experts in designing flows of dialogue that are automatically translated in a working bot on *DialoFlow* without writing a line of code. At the current stage, *Draw.io* is used to design the graph but no debug controls are implemented to help experts during the design process. For example, we could devise a strategy to indicate when the experts are not using the right colours or forget to complete a section in an intent rectangle. Moreover, future works could aim at formalizing the visual language to specify the right grammar to use for creating a flow of dialogue.

REFERENCES

[1] Valtolina, S., Barricelli, B.R., Dittrich,Y. 2012. Participatory knowledge-management design: a semiotic approach, JVLC. 23 (2), 103–115

[2] Shawar, A., Atwell, E., & Roberts, A. 2005. Faqchat as in information retrieval system. In Human language technologies as a challenge for computer science and linguistics (pp. 274-278)

[3] Braun, D., Hernandez-Mendez, A., Matthes, F., Langen, M.: Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In: 18th Annual SIGdial Meeting on Discourse and Dialogue (2017)

[4] Costabile, M. F., Fogli, D., Mussio, P., Piccinno, A.: Visual Interactive Systems for End-User Development: a Model-based Design Methodology. IEEE TSMCA, 37(6), pp. 1029--1046 (2007)

[5] Petre, M., Blackwell, A. F.: Children as Unwitting End-User Programmers. In: VL/HCC 2007, pp. 239-242

[6] Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In: End-User Development, pp. 1--8). Springer (2006)

[7] Nardi, B.: A Small Matter of Programming. MIT Press (1993)

[8] Brancheau, J. C., Brown, C. V.: The Management of End-User Computing: Status and Direction. ACM Computing Surveys, 25(5), pp. 437--482 (1993)

[9] Cypher, A.: Watch What I Do: Programming by Demonstration. The MIT Press (1993)