Dialogware - the "Software" for Conversational Agents: a Modular FrameNet-based Approach

Fabio Massimo Zanzotto, Gioele Minardi, Dario Onorati, Gabriele Cocino, Andrea Formichetti

Dept. of Enterprise Engineering, University of Rome Tor Vergata, Italy,
`fabio.massimo.zanzotto@uniroma2.it`

Abstract. Conversational agents are in every pocket where a smartphone is. It is the right time to foster a new generation of programmers to give a better personality to these conversational agents. In this paper we propose the *dialogware* as a novel form of software that should be produced by communication experts and artists: the *dialogware* programmers. We then introduce a modular FrameNet-based approach to dialogware programming along with a collaborative ecosystem for these new generation of programmers.

Keywords: Conversational Agents, Chatbots, FrameNet, Modular Programming

1 Introduction

Today, conversational agents are everywhere. Every pocket containing a smartphone hides a conversational agent ready to answer questions or to help with some everyday task. Conversational agents are not more confined in the realm of science fiction. They are definitely among us.

There is a wide range of stable techniques for building these astonishing conversational agents – a continuum from hand-crafting [12, 17, 7, 2, 3] to automatic induction from existing interactions [15, 13, 11, 19]. Hand-crafted converstional agents are considered relic of ancient past as automatically induced models generally obtain impressive performances. But, sometimes automatically induced conversational agents select extremely embarrassing answers and they can be hardly decoded. In fact, inspecting thought vectors (as Geoff Hinton calls these vectors [5]) of neural networks dialog systems can be a useless activity. On the contrary, hand-crafted conversational agents are clear enough to be easily controlled. If the planner fails, programmers can debug and find out what the problem is.

More important than the underlying technique, the key for the success of these conversational agents is their actual knowledge for reacting to stimuli. To have credible conversations, this knowledge should be produced or selected by experts of communication, experts in writing poems, novels or stories, and, finally, artists. In fact, these people are hired in conversational agent teams of big players, for example, to define Cortana, Microsoft has hired Jonathan Foster (a Hollywood film and TV writer) along with other people with the previous skills [9].

Hence, this is the dawn of the "*dialogware*" – a new form of software to program conversational machines. The "dialogware programmers" are the programmers for this new form of software. But, they are not programmers in the traditional sense and with the traditional software oriented mind. Hence, they need more intuitive interfaces.

In this paper, we propose the dialogware programming as a novel way of programming and we present an associated community-based ecosystem oriented to dialogware programmers. As any programming model, dialogware programming is based on modularity and reuse. To foster the development of dialogware modules, modules are organized using FrameNet [6, 4] and the dialogware programming ecosystem is equipped with easy-to-use and intuitive interfaces, allowing non-coders to easily contribute to the general dialogware library and to build their own conversational agents. We leverage on coding experience like Scratch [14], which are currently providing an option for non-programmers to code their own mobile apps [10] or program robotic environments [16].

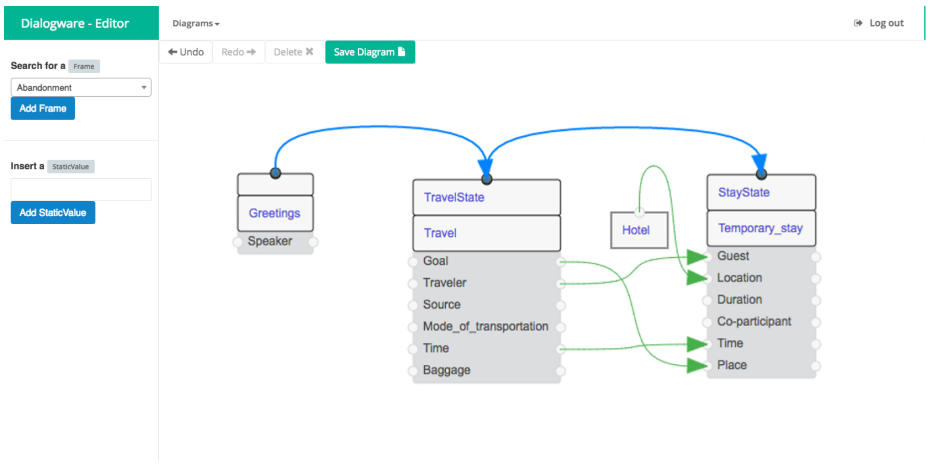


Fig. 1. Dialogware Diagram Editor

2 Dialogware: Programming Principles

Dialogware is the knowledge for "programming" conversational agents. It is a novel form of software. This section presents this model by introducing the modular approach, a basic hosting language and a way to produce dialog modules on the basic hosting language.

2.1 Building Conversational Agents by Combining Modules

The key point of our model is that conversational agents can be programmed with reusable dialog blocks and these dialog blocks can be organized using ontological languages and existing linguistic resources such as FrameNet [6, 4]. Hence, building conversational agents can be divided in two clear parts: combining dialog modules and programming specific modules if needed.

Having FrameNet [6, 4] and a set of ontological concepts, the activity of programming a new conversational agent for a specific task is the following. The first step is to select the prototypical situations, that is, the frames, involved in the dialog. In a goal based dialog, frame elements of each frame are considered as variables to be filled with a value through a conversation of the conversational agent of the user. The second step is to describe how dialogs over these frames interact in the general picture of the goal of the conversational agent. The last step is to program the specific dialogs for the specific frames if these are not still developed.

For example, we want to build a conversational agent for a travel agency. The goal of this agent is to gather the information needed to book the travel and the related accommodation. In the first step, we select the important frames which are *Travel* and *Temporary_stay* and we select additional dialogs such as the *Initial Greetings*. The *Travel* frame has all the frame elements needed to book a trip – *source*, *goal*, *mode_of_transportation*, *traveler* and *time* – and the *Temporary_stay* frame has all those to book an accommodation – *guest*, *duration*, *time*, *place* and *guest*. In the second step, we combine the frame dialog blocks to obtain the general dialog (see Fig. 1). The schema declares that the conversational agent starts from the *greetings*. Once this state is accomplished, the next dialog block is the *Travel* frame whose dialog aims to fill the free frame elements. The next state is the *Temporary_stay* frame and some of its frame elements are filled with values of the previous state, for example, the *goal* of *Travel* fills the *place* of *Temporary_stay*. In the last step, we fill the dialog blocks which are not defined.

2.2 Basic Hosting Language

To realize the strategy of building conversational agents by combining modules, we need a basic conversational agent technology that keeps tracks of the state of the conversation and has an explicit control of inner variables. These are the only real prerequisites for our approach.

Hence, we use the Artificial Intelligence Markup Language (AIML) [18] as host language. This language is based on a stimulus-response model where the interactions of conversational agents are described by prototypical stimuli associated to responses. Stimuli are written with a very simple pattern language which allows a wild card called *star* that can match sequences of characters. AIML has the definition of the status of the dialog (**topic**), the possibility of defining paraphrases of prototypical stimuli with the so-called *symbolic reduction* (**srai**), the possibility of introducing variability in the output with a random choice between alternative responses and, finally, the possibility of managing inner variables, that is, storing (**set**), retrieving (**get**) and checking (**condition**) values.

Although simple, AIML is enough versatile for hosting the first version of the dialogware programming paradigm.

2.3 Programming Basic Modules

Reusable goal-based dialog modules aims to fill a set of variables by asking questions. These modules based on FrameNet and on ontological resources are extremely interesting when building a goal-based conversational agent. As these modules are one of the key point of our model, we present a way to program these modules in a simple stimulus-response hosting language.

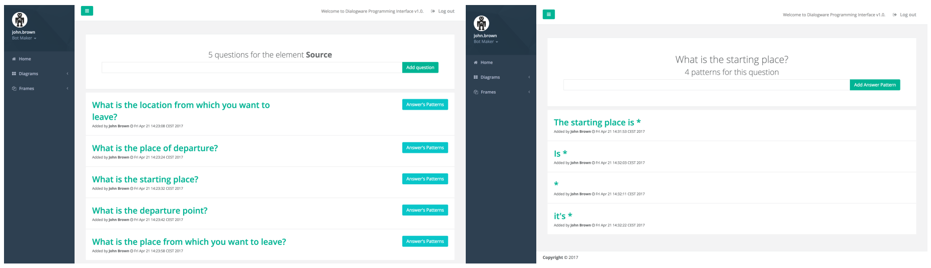
In AIML, we realize these reusable goal-based dialog modules with three main components: (1) a controlling backbone; (2) a set of interaction generators to stimulate answers containing values for variables; and, (3) a set of answer interpreters that capture values for variables.

The dialog module conversation backbone aims to fill all the variables of a given frame. Then, the backbone controls which variable has still to be filled and initiates the dialog for stimulating answers to fill missing variables. This backbone is then called when the state is entered and at the end of each answer interpretation. Given a frame with the variables v_1, v_2, $\ldots$, v_n, the backbone is a stimulus-response pair with the following aspect in a pseudo-language:

```
stimulus: $STARTPOINT$
response:
  cases:
    $v_1=\emptyset$?: call: $Request\_v_1$
    $v_2=\emptyset$?: call: $Request\_v_2$
    ...
    $v_n=\emptyset$?: call: $Request\_v_n$
    $default$: call: $FRAMEDONE$
```

where $\emptyset$ is the empty value for a variable. This controlling backbone is called by the stimulus $STARTPOINT$ and calls $Request\_v_i$ for the first variable v_i which is still empty. $Request\_v_i$ is the set of interaction generators that stimulates answers for filling variable v_i. When all the variables are filled with values, the backbone calls $FRAMEDONE$. For example, for the frame *Travel*, variables are *Traveller*, *Source*, *Goal* and so on.

The set of interaction generators $Request\_v_i$ have the following form:



5 questions for the element **Source**

What is the location from which you want to leave?
Added by **John Brown** Fri Apr 21 14:23:08 CEST 2017

What is the place of departure?
Added by **John Brown** Fri Apr 21 14:23:24 CEST 2017

What is the starting place?
Added by **John Brown** Fri Apr 21 14:23:32 CEST 2017

What is the departure point?
Added by **John Brown** Fri Apr 21 14:23:42 CEST 2017

What is the place from which you want to leave?
Added by **John Brown** Fri Apr 21 14:23:58 CEST 2017

What is the starting place?
4 patterns for this question

The starting place is *
Added by **John Brown** Fri Apr 21 14:31:53 CEST 2017

Is *
Added by **John Brown** Fri Apr 21 14:32:03 CEST 2017

*
Added by **John Brown** Fri Apr 21 14:32:11 CEST 2017

it's *
Added by **John Brown** Fri Apr 21 14:32:22 CEST 2017

Fig. 2. Dialogware Frame Editing

```
stimulus: $Request\_v_i$
response:
  random choice:
    "[$Request\_v_i$ interaction 1]"
    "[$Request\_v_i$ interaction 2]"
    ...
    "[$Request\_v_i$ interaction m]"
```

where "[$Request\_v_i$ interaction j]" is an actual utterance for asking to fill the specific variable v_i. For example, for the variable *Goal* in the frame *Travel*, possible interactions are *"Where are you going?"* or *"What's your final destination?"*.

For each request $Request\_v_i$, there is a set of answer interpreters that are in charge of extracting variable fillers from answers given by users. These answer interpreters have the following form:

```
stimulus: [A possible answer to $Request\_v_i$ with a variable filler in *]
response:
  set $v_i$ to : value that is filling *
  call: $STARTPOINT$
```

Each answer interpreter captures a form of answer for a specific request $Request\_v_i$, extracts the value for the variable with a wildcard $*$, fills the variable v_i and, finally, call back the starting point of the dialog module. For example, a possible answer to $Request\_v_i$ for the variable *Goal* is *"I want to go to *"*. Then, interactions like *"I want to go to New York"* will be matched and the value of $*$ will be *New York*. The value of $*$ will be used to fill the variable *Goal*.

3 Dialogware: the Programming Eco-system

The Dialogware programming eco-system consists of two main parts: the Dialog Diagram Editor and the Frame Editing. This section describes these two main components and the overall implementation details of the ecosystem.

3.1 Diagram Editor

The Dialogware Diagram Editor is the core of the programming ecosystem. It offers a graphical interface for the definition of creating the control diagram of a goal-oriented conversational agent. The diagram editor is composed of two main section: the sidebar and the actual canvas (see Figure 1). The sidebar contains the palette of elements that can be added in the control diagram: frames and static values. When a user clicks on the *Add Frame* button, the selected frame is imported into the canvas. Then, the application prompts the user to specify the *status* of the dialogue block in the overall diagram. The main canvas allows to define the diagram by creating transitions among blocks (blue solid arrows) and declaring how variables are filled from one frame to the other (green thin arrows).

3.2 Frame Editing

The Frame Editing section is used to add interactions associated with the elements of a frame, as well as to add the response pattern to them. The frame editing allows to select the frame and to work on the interactions of the selected frame. By selecting a frame and an element the Frame editor shows the list of all questions associated with the selected frame element (see Figure 2 left panel). Finally, by selecting the *Answer's Patterns* button, the frame editor shows all the response patterns associated to the selected question. The Frame editor allows to write both novel questions and novel possible answers from different users (see Figure 2 right panel).

4 Conclusion and Future Work

Conversational agents are a rapidly expanding market. We have presented an approach to develop reusable dialog modules by introducing the dialogware as a novel form of software. Splitting dialogware programming in blocks is undoubtedly an effective approach in terms of software development, as well as being an effective method to reduce the overall complexity of building knowledge for conversational agents. As shown also for serious games [8], FrameNet [6, 4] is a very important source for organizing programming in general and the production of dialogware modules in particular.

The community of practitioners of conversational agents is rapidly increasing. Our dialogware programming ecosystem wants to foster a revolution: transforming artists, experts of communication and simple users of conversational agents in developers of dialogware.

In the future, by leveraging on techniques of semantic textual similarity [1, 20], we will improve our dialogware programming ecosystem by introducing modules that expand interactions. This will speed up the manual and controlled production of dialogware programs.

References

1. Agirre, E., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W.: *sem 2013 shared task: Semantic textual similarity. In: Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity. pp. 32–43. Association for Computational Linguistics, Atlanta, Georgia, USA (June 2013), `http://www.aclweb.org/anthology/S13-1004`
2. Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: An Architecture for a Generic Dialogue Shell. Nat. Lang. Eng. 6(3-4), 213–228 (2000), `http://dx.doi.org/10.1017/S135132490000245X`
3. Augello, A., Gentile, M., Weideveld, L., Dignum, F.: A Model of a Social Chatbot, pp. 637–647. Springer International Publishing, Cham (2016), `https://doi.org/10.1007/978-3-319-39345-2_57`
4. Baker, C.F., Fillmore, C.J., Lowe, J.B.: The berkeley framenet project. In: Proceedings of Proceedings of COLING-ACL. Montreal, Canada (1998)
5. Corrado, G.: Computer, respond to this email (2015), `https://research.googleblog.com/2015/11/computer-respond-to-this-email.html`
6. Fillmore, C.J.: The case for case. In: Bach, Harms (eds.) Universals in Linguistic Theory., pp. 1–88. New York: Holt, Rinehart, and Winston (1968)
7. Freedman, R.: Plan-based Dialogue Management in a Physics Tutor. In: Proceedings of the Sixth Conference on Applied Natural Language Processing. pp. 52–59. No. 9720359 in ANLC '00, Association for Computational Linguistics, Stroudsburg, PA, USA (2000), `http://dx.doi.org/10.3115/974147.974155`
8. Gentile, M., Città, G., Ottaviano, S., La Guardia, D., Dal Grande, V., Allegra, M., Jarvinen, A.: A Semantic Frame Approach to Support Serious Game Design, pp. 246–256. Springer International Publishing, Cham (2016), `https://doi.org/10.1007/978-3-319-50182-6_22`
9. Goldman, D.: How Cortana got her corny jokes (2015), `http://money.cnn.com/2015/07/30/technology/windows10-microsoft-cortana/{\%}0Ahttps://chatbotconf.com/{\%}0A`
10. Gray, J., Abelson, H., Wolber, D., Friend, M.: Teaching CS Principles with App Inventor. In: Proceedings of the 50th Annual Southeast Regional Conference. pp. 405–406. ACM-SE '12, ACM, New York, NY, USA (2012), `http://doi.acm.org/10.1145/2184512.2184628`
11. Henderson, M., Thomson, B., Young, S.: Deep Neural Network Approach for the Dialog State Tracking Challenge. In: Proceedings of the SIGDIAL 2013 Conference. p. 467{\textendash}471. Association for Computational Linguistics, Association for Computational Linguistics, Metz, France (2013), `http://www.aclweb.org/anthology/W13-4073`
12. Larsson, S., Traum, D.: Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. Journal of Natural Language Engineering 6(3-4), 323–340 (2000)
13. Lison, P.: Structured Probabilistic Modelling for Dialogue Management Doctoral Dissertation by. Ph.D. thesis, University of Oslo (2013)
14. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M.: Scratch: A Sneak Preview. In: Proceedings of the Second International Conference on Creating, Connecting and Collaborating Through Computing. pp. 104–109. C5 '04, IEEE Computer Society, Washington, DC, USA (2004), `http://dx.doi.org/10.1109/C5.2004.33`

15. Roy, N., Pineau, J., Thrun, S.: Spoken Dialogue Management Using Probabilistic Reasoning. In: 38th Annual Meeting of the Association for Computational Linguistics. No. Acl, Hong Kong, China, `http://www.aclweb.org/anthology/P00-1013`
16. Rusk, N., Berg, R.: New Pathways into Robotics : Strategies for Broadening Participation. Journal of Science Education and Technology 17(1), 59–69 (2008)
17. Seneff, S., Polifroni, J.: Dialogue Management in the Mercury Flight Reservation System. In: Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3. pp. 11–16. ANLP/NAACL-ConvSyst '00, Association for Computational Linguistics, Stroudsburg, PA, USA (2000), `http://dx.doi.org/10.3115/1117562.1117565`
18. Wallace, R.S.: The Anatomy of A.L.I.C.E., pp. 181–210. Springer Netherlands, Dordrecht (2009), `http://dx.doi.org/10.1007/978-1-4020-6710-5\_13`
19. Williams, J.D., Zweig, G.: End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning (2016)
20. Zanzotto, F., Dell'Arciprete, L.: Distributed tree kernels. In: Proceedings of International Conference on Machine Learning. pp. 193–200 (2012), `http://www.scopus.com/inward/record.url?eid=2-s2.0-84867126965&partnerID=40&md5=0d51c0ed7070baf730f887c818a8c177`