A Systematic Approach to Implementing Chatbots in Organizations – RTU Leo Showcase

Antons Mislevics[1], Janis Grundspenkis[2] and Raita Rollande[3]

[1] Riga Technical University, Riga, Latvia, antons.mislevics@rtu.lv
[2] Riga Technical University, Riga, Latvia, janis.grundpsenkis@rtu.lv
[3] Ventspils University of Applied Sciences, Ventspils, Latvia, raita.rollande@venta.lv

Abstract. *Leo* is a chatbot that was successfully developed to support potential new students during the admission process. While developing *Leo* the authors have concluded that existing publications do not cover the approach of developing virtual assistants or chatbots in organizations in enough details. This paper is aimed to fill-in this gap by describing a systematic approach that was followed by RTU team while developing *Leo*.

1 Introduction

In early 2018 Department of Public Affairs of Riga Technical University (RTU) initiated a project with a goal to develop a virtual assistant *Leo* [3]. The idea was that such virtual assistant could improve communication and reduce an effort required for providing information to existing and potential students. This project was also seen as a great opportunity for RTU students to get practical experience in the modern field of virtual assistants.

In the beginning of the project, the team had conducted an analysis of existing literature and publications in order to understand how such projects can be approached. It was observed, that also the field of virtual assistants and chatbots is getting a lot of attention in recent years, available materials can be logically organized into three categories:

1) business/marketing articles describing why virtual assistants are better than other technologies and how these can be applied;
2) technical articles describing how to implement virtual assistants using specific software product/platform;
3) scientific research publications describing how to apply virtual assistants in specific scenarios, or how to improve specific aspects of such solutions.

As a result, the authors have concluded that existing publications do not cover the approach of developing virtual assistants or chatbots in organizations in enough details. In this paper we aim to fill-in this gap by describing a systematic approach that was followed by RTU team while developing virtual assistant *Leo*. The authors believe that similar approach can be followed to develop chatbots in different domains as well.

The term “virtual assistant” is broadly used to describe a software agent that is helping users to achieve specific goals via human-like communication. Such communication can be performed via voice (spoken communication) or text (text-based communication) [2]. The term “chatbot” is used to describe a virtual assistant that is communicating via text. In this paper the term “chatbot” is used, since it better describes the way *Leo* was built. However, since different types of virtual assistants have many similarities [2], the systematic approach described in this paper is applicable to implementing other types as well.

2 Chatbots

Chatbots are software agents that communicate with end-users via text-based conversations. Depending on the nature of conversations, chatbots are divided into open- and closed-domain [2, 4]. Open-domain chatbots can participate in a free form conversation with a user, having no specific goal defined. Closed-domain chatbots are built for helping a user to achieve specific goal. The approach described in this paper is used to implement closed-domain chatbots.

In contrast to desktop, mobile and web applications, chatbots do not provide graphical user interface (GUI). Instead, users are communicating with chatbots via conversational user interface (CUI) or integration with existing messaging applications. Switching from GUI to CUI has several advantages [4]:

- users do not need to download and install specific application;
- in order to access chatbot functionality, users do not need to launch specific application – they can continue using messaging application, which they are also using for other conversations;
- it is possible to identify users via existing messenger account, without asking them to register;
- implementing CUI takes less time and is cheaper;
- CUIs are intuitive and do not require specific interface learning;
- updating or adding new functionality to CUI is easy – there is no need to develop a separate application per service.

Based on the approach how the conversation is handled, CUIs can be categorized into flow-based and intent-based modes [12, 13, 14]. In flow-based mode a chatbot is gradually moving a user towards the right answer. Such bots often provide options for possible answers to the user. Based on the option selected, conversation flow progresses to the next state. In intent-based mode the user can enter a message in a free text form. The chatbot is analyzing received text and tries to understand what is the intent of the user. The answer is provided based on the identified intent. Intent-based mode can be further split into selective and generative implementations [9]. In selective scenario the bot has a knowledge base of all possible answers. When answering a question, the bot selects one of the answers from this knowledge base. In generative scenario the bot does not have a knowledge base and thus is generating full text of the response.

Leo has both flow- and intent-based conversations. Selective approach is used for intent-based mode.

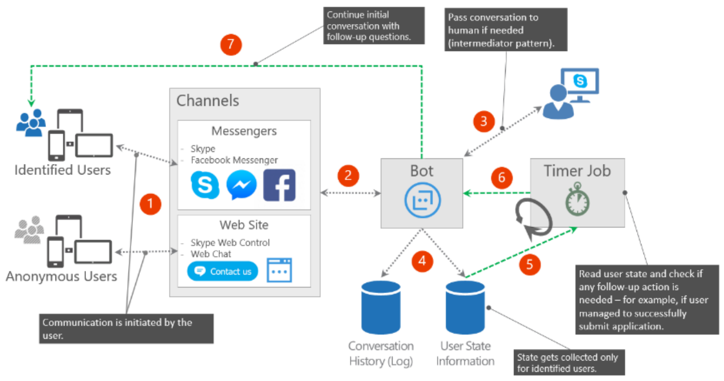
Continue initial conversation with follow-up questions.
Pass conversation to human if needed (intermediator pattern).
7
Channels
Messengers
- Skype
- Facebook Messenger
Identified Users
1
2
3
Bot
Timer Job
6
Web Site
- Skype Web Control
- Web Chat
Contact us
Anonymous Users
4
5
Read user state and check if any follow-up action is needed – for example, if user managed to successfully submit application.
Communication is initiated by the user.
Conversation History (Log)
User State Information
State gets collected only for identified users.

Fig. 1. Overall architecture of a chatbot solution

Figure 1 shows the overall architecture of a chatbot solution, that was defined when implementing *Leo*. The following interactions are happening in the system:

1) A user initiates new conversation with a chatbot via one of supported channels. Depending on the channel used, a user will be identified or anonymous. Identified users come via channels that require some form of authentication – for example, Skype [8] or Facebook Messenger [10]. In contrast, anonymous users come from channels that do not require authentication – for example, chat control on a web site. This distinction is important, because identified users have persistent conversation sessions, what allows the chatbot to continue original conversation later – for example, in order to ask some follow-up questions.

2) Conversations coming via different channels are handled by a single implementation of a chatbot.

3) In scenarios where the chatbot cannot handle conversation by itself, it can pass conversation to human operator. Intermediator pattern allows to make this transition transparent – the chatbot submits a question received from the user to the operator by creating a new conversation session, receives operator's response, and sends it back to the user via original conversation session. This way identity of the operator is not exposed and the user does not have to switch conversation context.

4) The chatbot collects the following information during conversations: a) full log with histories of conversations (questions and answers); b) the state of the conversation session for each identified user. Conversation history log is used to analyze chatbot performance and improve it over time. User state information can be used to initiate follow-up actions with identified users.

5) Background timer job is regularly reading collected user state information and checking if any follow-up actions are needed for specific users. For example, based on the last conversation, the user was planning to apply for some service. The chatbot could initiate a follow-up conversation to ask if there were any problems during the application process.

6) If any follow-up actions are needed, timer job notifies the chatbot that follow-up conversation must be started.

7) The chatbot continues original conversation with identified user by sending follow-up questions.

3 Implementation Approach

When implementing *Leo* the team had defined a systematic approach, that can be followed to develop other chatbots as well. This section describes every step of this approach.

Step 1: Agree on design principles

First, the team needs to agree on the main design principles. For *Leo* we've agreed on the following:

1) It must be clear to all users that *Leo* is a bot and not a human.

2) *Leo* must be easily understandable to users.

3) The team must focus on business goal and avoid unnecessary overcomplication.

Step 2: Define business goal, communication channels and content areas

It was agreed that initially *Leo* will support potential new students during the admission process. The main goal is to reduce the number of calls received by admission commission operators during admission period. The first implementation will work on Latvian language only.

By analyzing information on conversations handled by the commission during previous years, the following content areas have been identified:

1) Where to study – information about RTU study programs, and simple career test to find out which study programs match interests of specific person;

2) How to apply – information about admission process, required documents and deadlines;

3) Get answer to common question – answers to free text questions.

Analysis of target audience allowed to identify the following target communication channels: Web chat control on RTU homepage, Facebook (via web or messenger application) and Skype.

Chatbot CUI was design the way, that user experience in all of these channels is consistent.

Step 3: Prepare content and define conversation approach for each content block

The following content blocks were prepared:

1) Information about RTU study programs;

2) Questions for career test with response options;

3) Table mapping career test responses to recommended study programs;

4) Information about admission process;

5) Knowledge base of frequently asked questions: table with questions and answers; each answer can be associated with multiple questions, which show different ways to ask for the same information.

Since information for content blocks 1-4 is well structured, it was decided to organize it in a flow-based conversation. For these blocks *Leo* is gradually moving the user to relevant content. From CUI perspective, *Leo* is asking questions and proposing possible response options. For each response option an action button is provided, what allows the user to click on the button, or type-in desired option in a text form. This follows design principles 2 and 3. This also supports design principle 1, since the chatbot does not pretend that he knows everything, and he explicitly guides the user through his knowledgebase.

For content block 5 intent-based dialog is used. The user can ask the question, and *Leo* will return the most appropriate answer from the knowledge base. In order to follow design principle 1: a) before proposing to ask a question, *Leo* describes that he does not know all the answers, and will select the most appropriate answer from the knowledgebase; b) after returning an answer, *Leo* asks if this was helpful, and if not, proposes to rephrase the questions or contact admission commission operator.

Step 4: Design visual identity

Visual identity is important, because it affects how users perceive a chatbot. The team went through multiple brainstorming rounds to agree on:

1) The name of the bot;
2) Short introduction message/story for the bot;
3) Bot logo/icon;
4) Color scheme in chat dialogs;
5) Overall style of the language, what the bot will use in conversations.

It should also be validated that the bot looks and interacts similarly in all of the target communication channels.

Step 5: Chatbot implementation

Implementation was organized in several parallel streams:

1) The overall platform and integrations with target communication channels;
2) Flow-based dialogs;
3) Intent-based dialog for question answering block;
4) Custom web control, for integration into RTU webpage;
5) Logging mechanism that can be used to analyze conversations held by the chatbot.

Implementation activities for each stream were performed in iterations. This allowed to gather feedback and adjust *Leo* accordingly. For more information on technical implementation of *Leo*, see section 4 Technical Details.

Step 6: User testing

User testing was performed on intermediary results for every stream. Testing was performed by RTU employees (subject matter experts), RTU students and secondary school graduates. Testers were asked to handle conversations with *Leo* and provide feedback describing their experience. Specifically, we were interested in:

1) If their experience is intuitive overall?
2) If flow-based dialogs are intuitive and help to get necessary information?
3) If the knowledge base of frequently asked questions is missing answers to some important questions?
4) If their questions are answered accurately in question answering block?

Step 7: Implementing improvements

The feedback received from user testing was analyzed and incorporated into *Leo*. Some examples of what such feedback can affect are listed below:

1) Changes in wordings of messages returned by the bot;
2) Adjustments in flows of flow-based dialogs;
3) Modifications in texts on action buttons in flow-based dialogs;
4) New answers in the knowledge base of frequently asked questions.

In addition, information collected in conversation log was analyzed. It helped to understand how conversations were progressing overall, and also to improve question answering block. First, it was used to better evaluate bot performance, by analyzing the full list of questions received by the bot, and how these were answered. Second, it helped to enrich training/validation set of questions and answers used to train *Leo*. Having larger training/validation set allowed to improve the accuracy of question answering model.

It should be noted, that evaluation of accuracy for question answering models is a non-trivial task, that has been a topic for active research over the recent years. In case of *Leo* such evaluation was performed based on subjective feedback received from users.

Step 8: Launch in production

Production launch of the chatbot involved several activities:

1) Deploying and configuring technical implementation of the chatbot;
2) Integrating chatbot web control into the web page;
3) Registering the chatbot on every target channel;
4) Getting people to know about the new chatbot by preparing and distributing informational materials.

It should be noted, that different channels have different requirements for bots to be activated and available for communication. Usually this involves some form of a technical review. It is important to start registration process in advance, since it may take up to several weeks, until the chatbot will become available to users of specific channel.

After the launch of *Leo* we've started reviewing conversation log on a regular basis in order to understand how the bot is used and what changes might be needed. This process is similar to the one described in Step 7.

4 Technical Details

This section provides more details on technical implementation of *Leo*. The goal of this section is to keep descriptions on a high-level, just to give readers the overall understanding of how *Leo* is built.

4.1 Overall Technical Design

Figure 2 shows the overall technical design of *Leo*. Microsoft Azure Bot Service [5, 6] is selected as a technology, that provides an interface to target communication channels. It passes messages (1) received from users to Bot Web Application, that was developed using Microsoft Bot Builder SDK [7]. Bot Web Application is used to implement flow-based dialogs. Free text questions are passed (4) to QnA Web Application. This web application hosts machine learning model, that was trained to return answers from the knowledge base. Bot Web Application writes (2) the history of all conversations into the conversation log. This log is stored in Azure Table. In order to view and analyze log information, Leo administrators can add (3) this Azure Table as a data source to Excel worksheet.

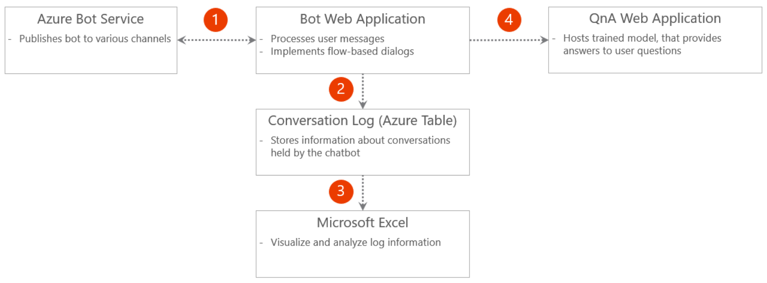
Azure Bot Service
- Publishes bot to various channels
1
Bot Web Application
- Processes user messages
- Implements flow-based dialogs
4
QnA Web Application
- Hosts trained model, that provides answers to user questions
2
Conversation Log (Azure Table)
- Stores information about conversations held by the chatbot
3
Microsoft Excel
- Visualize and analyze log information

Fig. 2. *Leo* technical design

4.2 Flow-Based Dialogs

Flow-based conversation blocks are implemented in a form of a state machine. Each state is described by the message that is sent by the bot. Transitions from one state to another describe expected response options. Each transition is defined by the text that will appear on the action button and a set of regular expressions that are used to match plain text user input to specific option. This is required if instead of clicking the button, the user decides to input response text manually. If the bot cannot find a match between text response and any response options, he will ask to resubmit the answer, and show action buttons to make input simpler.

In order to address all identified scenarios for flow-based dialogs, several types of states where implemented:

1) Text message – text message with optional markdown formatting;

2) Hero Card message – advanced message with image and description text;

3) Carousel message – advanced message with multiple images and descriptions, that can be switched left and right as a carousel;

4) Delay message – is used to simulate "thinking time" of the bot. The bot waits for specified delay time before proceeding to the next state;

5) Call external web service – passes received text message to external web service and returns received response as a message to the user;

6) Custom message – implements specific state with custom logics: for example, analyze career test results and show relevant study programs.

All states that return a response can also define action buttons for response options. However, it is also possible to define that specific state expects a free text input from the user.

4.3 Intent-Based Dialogs

Intent-based dialogs are user to implement question answering block. Technically, when the user is asking a question, this message is received by *Call external web service* stage and gets passed to QnA Web Application. QnA Web Application exposes a trained machine learning model via a web service.

Technically machine learning model is developed on Python using gensim [18], fastText [19], TensorFlow [16] and Keras [17]. The model is wrapped into Flask [20] web application that exposes it as a web service.

The model is trained on a knowledge base of questions and answers. Each answer can be associated with multiple questions, which are different ways to ask for the same information. In order to provide better support for language constructs, texts are first transformed to vectors of word vectors. Word vectors are generated by using pre-trained fastText word vectors for Latvian language [11, 10].

The model itself is trained as a Siamese neural network, where during training time combinations of question/right answer and question/wrong answer are shown to the model. The goal is to make the model "learn" that questions and right answers are similar (located close in latent space), but questions and wrong answers are different (the distance in latent space is large).

During prediction phase the model receives a question, gets it "position" in latent space, and compares it to positions of all answers from the knowledge base. The answer having shortest distance from question is returned as a correct one.

4.4 Conversation Log

Information about every message received or sent by the bot is recorded in a conversation log. The following information is captured:

1) *Event Time* – time when the event had occurred;
2) *Channel Id* – channel over which the conversation was performed;
3) *From Id / From Name* – user with whom the conversation was performed;
4) *Conversation Id* – unique id of the conversation; can be used to identify messages sent in a single conversation, and also repeating conversations, when identified user decided to start new conversation immediately after completing previous one, or over time;
5) *Message Text* – the text of the message sent or received;
6) *Dialog Id* – id of the state at which the message was sent or received.

Conversation log is used to analyze chatbot performance and implement improvements – see Step 7 in Section 3 Implementation Approach. In addition, it also helps to collect statistics on bot usage – for example, unique or repeating conversations via every channel.

5 Conclusions

Leo is a chatbot that was developed to support potential new students during the admission process. Based on the learnings from Leo implementation, the authors have defined an approach that allows to develop chatbots in organizations in a systematic way. Proposed approach consists of eight steps and describes the full lifecycle of chatbot implementation project starting from definition of architecture design principles and analysis of business scenarios, up to the launch in production. It also highlights the importance of implementing continuous improvements of the chatbot, and describes how this can be done by collecting and analyzing conversation logs. The last section provides an overview of a technical implementation of Leo, to give readers the overall understanding of how *Leo* is built.

The systematic approach described in this paper is not specific to text-based chatbots in education. It is applicable to other types of virtual assistants in other domains as well.

References

1. A. Fadhil. Domain Specific Design Patterns: Designing For Conversational User Interfaces, arXiv preprint arXiv:1802.09055, 2018. Available: https://arxiv.org/ftp/arxiv/papers/1802/1802.09055.pdf
2. V. Ilievski. Building Advanced Dialogue Managers for Goal-Oriented Dialogue Systems, arXiv:1806.00780 [cs], Jun. 2018. Available: https://arxiv.org/pdf/1806.00780.pdf
3. Riga Technical University, Virtual Assistant Leo (Latvian). Available: https://www.rtu.lv/lv/virtualais-asistents-leo
4. M. Hardalov, I. Koychev, and P. Nakov. Towards Automated Customer Support, arXiv:1809.00303 [cs], vol. 11089, pp. 48–59, 2018. Available: https://arxiv.org/pdf/1809.00303.pdf
5. Microsoft. Azure Bot Service. Available: https://azure.microsoft.com/en-us/services/bot-service/
6. Microsoft. Azure Bot Service Documentation. Available: https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-3.0
7. Microsoft. Bot Builder SDK. Available: https://github.com/Microsoft/BotBuilder
8. Microsoft. Skype. Available: https://www.skype.com
9. A. Tammewar, M. Pamecha, C. Jain, A. Nagvenkar, and K. Modi. Production Ready Chatbots: Generate if not Retrieve, Nov. 2017. Available: https://arxiv.org/pdf/1711.09684.pdf
10. Facebook Research. Library for fast text representation and classification: Pre-trained word vectors. Available: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
11. P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching Word Vectors with Subword Information, arXiv:1607.04606 [cs], Jul. 2016. Available: https://arxiv.org/abs/1607.04606
12. Z. Yan, N. Duan, P. Chen, M. Zhou, J. Zhou, and Z. Li. Building Task-Oriented Dialogue Systems for Online Shopping, 2017. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/download/14261/13975
13. Z. Yin, K. Chang, and R. Zhang. DeepProbe: Information Directed Sequence Understanding and Chatbot Design via Recurrent Neural Networks, Jul. 2017. Available: https://arxiv.org/abs/1707.05470
14. M. G. de Bayser, P. Cavalin, R. Souza, A. Braz, H. Candello, C. Pinhanez, J.-P. Briot. A Hybrid Architecture for Multi-Party Conversational Systems, May 2017. Available: https://arxiv.org/pdf/1705.01214.pdf
15. Facebook. Messenger: https://www.messenger.com

16. Tensorflow. Available: https://www.tensorflow.org/
17. Keras. Keras Documentation. Available: https://keras.io/
18. R. Rehurek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks: 45-50, May 2010. Available: https://radimrehurek.com/gensim/about.html
19. fastText. Available: https://fasttext.cc/
20. Flask. Available: http://flask.pocoo.org/