Choosing a chatbot development tool

Sara Pérez-Soler, Sandra Juárez-Puerta, Esther Guerra, Juan de Lara
Modelling and Software Engineering Research Group
http://miso.es
Computer Science Department
Universidad Autónoma de Madrid (Spain)

Abstract. *Chatbots are programs that supply services to users via conversation in natural language, acting as virtual assistants within social networks or web applications. Companies like Google, IBM, Microsoft or Amazon have released chatbot development tools with different functionalities, capabilities, approaches and pricing models. With so many options, companies that want to offer services through chatbots can find choosing the right tool difficult. To help them make an informed choice, we review the most representative chatbot development tools with a focus on technical and managerial aspects.*

Keywords: Software Engineering, Chatbots, Natural Language Processing

1. Introduction

Chatbots are programs with a conversational user interface. Their popularity is rising because they enable accessing all sorts of services (e.g., booking flights, checking weather conditions) from web applications or social networks like Telegram, Twitter, Skype or Slack. This way, users can access those services without installing new apps and interacting with the service is simplified by the use of natural language (NL) [5].

Many companies are developing chatbots to automate customer support and provide ubiquitous access to the company services. At the same time, plenty of platforms and frameworks have emerged to ease chatbot construction. Large software companies like Google, Microsoft, IBM or Amazon have created chatbot development platforms, but many other alternatives exist. These platforms provide diverse functionality regarding natural language processing (NLP), the structure of the conversation flow, the ability to connect the chatbot to existing information systems, or the support for testing and deployment.

Choosing the best chatbot development tool for a particular need is difficult. Making an incorrect tool decision may lead to non-compliance with chatbot technical requirements or with software development company policies. Some websites and informal blogs compare some available options to build chatbots [1, 2, 3, 4], and researchers have identified aspects to consider in chatbot design (functional, integration, analytics and quality assurance) [8]. Instead, we analyse technical and managerial factors of the most representative chatbot creation tools, to help developers and managers in making informed choices on the optimal tools for their interest. This analysis can be used as a reading grid to select a tool based on technical criteria (e.g., "*we need a chatbot to access our current information system by text and voice, in both English and Spanish,*") and managerial constraints (e.g., "*my developers lack experience in developing chatbots, we do not have the capacity to deploy on-premises, and we are already using Amazon cloud*").

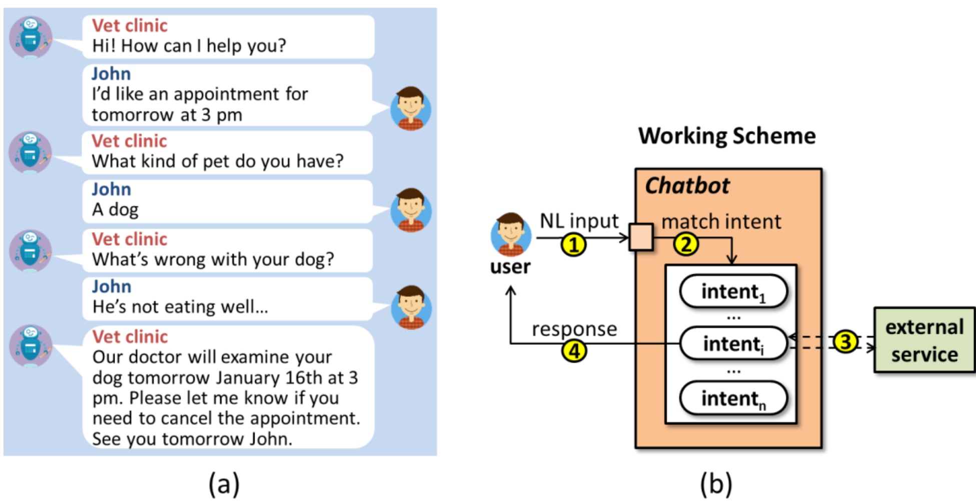

Figure 1: (a) Example of user interaction. (b) Working scheme of a chatbot.

2. What's in a chatbot?

A chatbot is a program supporting user interaction via conversation in NL, and normally accessible through the web or social networks. As an example, assume that a vet clinic has an information system with a database storing information about veterinarians and appointments, and decides to bring its services closer to customers by means of a chatbot to which customers can ask about opening hours and make appointments. This chatbot would allow the clinic to offer 24/7 service, reduce costs (e.g., decreasing customer telephone calls) and widen the potential customers. Figure 1(a) shows an example of a user interacting with the envisioned chatbot.

As Figure 1(b) shows, a chatbot is organized around intents that represent possible user's intentions and permit accessing the offered services. These intents typically reflect use cases of the chatbot. As an example, the chatbot for the clinic would define two intents: one to inform about opening hours, and another for making appointments. Upon receiving a user input in NL (label 1 in the figure), the chatbot identifies the matching intent (label 2). Depending on the intent, the chatbot may need to access external services, like the clinic database if the intent is setting an appointment (label 3). Finally, the chatbot replies to the user, e.g., confirming the appointment (label 4).

Figure 2(a) shows a process diagram with the main activities that designing a chatbot entails. The development process is not necessarily linear, but often requires iteration. Moreover, activities like validation and testing are needed throughout the process. Figure 2(b) contains a structural diagram (a UML class diagram) with the constituent elements of a chatbot. The numbers in this diagram identify the process step where the elements are defined.

First, developers must identify the *intents* that the chatbot will handle. While traditional applications typically offer their functionality via graphical interfaces, chatbots expose it through conversation. To match the intent corresponding to a user input phrase, developers can resort to NLP libraries – like the *Stanford Parser* [6] or the *Natural Language Toolkit* (NLTK) [7] – as they permit analysing the phrase structure and provide facilities for tokenizing and part-of-speech tagging, among others. This gives unlimited flexibility regarding the NL structure, but the implementation is costly. Hence, for narrow domains (like our clinic), it is simpler to train the chatbot with *training phrases* (i.e., examples of expected

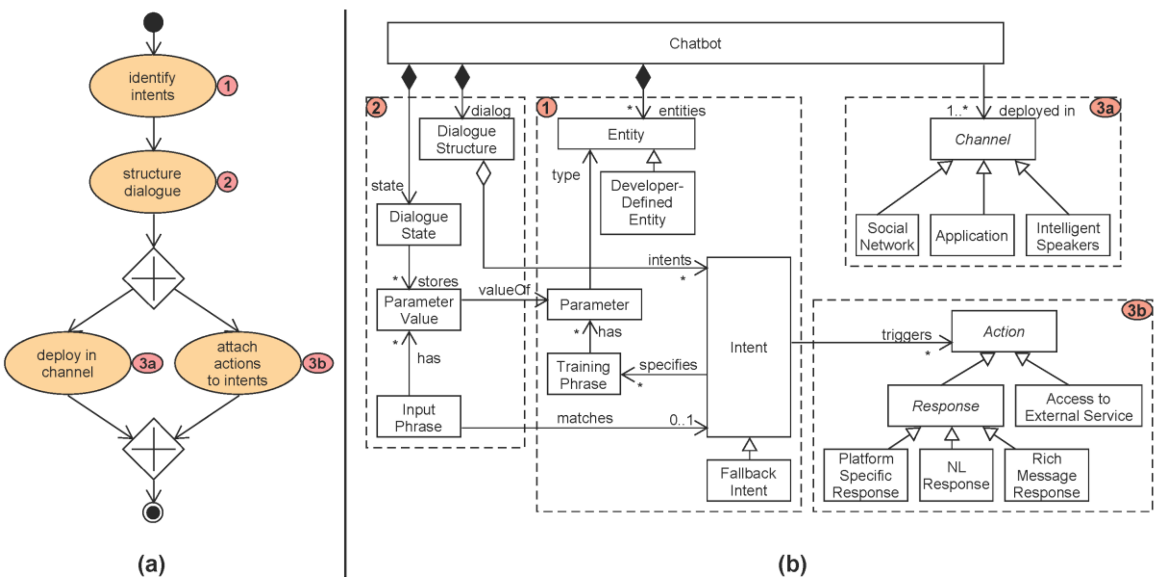

Figure 2: (a) Process diagram for chatbot design. (b) Structural diagram of chatbot concepts.

phrases) characterizing each intent. We can find libraries and services that apply machine learning for this purpose, like Microsoft's *LUIS* (https://luis.ai) or *Rasa NLU* (https://rasa.com). These libraries also support extracting *parameters* from phrases. A parameter is a piece of relevant information that needs to be extracted from a phrase, such as the date of an appointment. Parameters are conformant to a given *entity* type. Most chatbot development tools provide predefined entities (e.g., dates, numbers) and developers can define new ones (e.g., pet types). In addition, chatbots may define *fall-back intents,* used when the chatbot does not recognize the user utterance.

Besides intents, developers need to define the *dialogue structure* to accomplish a task. For example, after the user requests an appointment in Figure 1(a), the chatbot asks the kind of pet and problem, and only then the appointment is fixed. For this purpose, the chatbot needs to store the *dialogue state* – often in so-called *contexts* – to carry the information of previous *input phrases* through the stages of the conversation.

Moreover, developers need to identify the *actions* that each intent triggers. These may comprise invocations to *external services*, and include the chatbot *response* either in NL or using rich messages or mechanisms specific to the deployment platform. In our example, the chatbot needs to access the clinic information system to check for available slots and set appointments, and replies with the appointed date and time.

Finally, developers must deploy the chatbot in some *channel*. Typical channels are social networks, websites, or smart speakers like *Amazon Echo* or *Google Home*. In addition to NL, each channel may support specific interaction possibilities that can be exploited to obtain effective chatbots. For instance, to prompt the user to select among a small set of options (e.g., the available appointment slots within one day), presenting each option as a button can be less error-prone. However, different channels may support distinct interaction mechanisms. For example, Telegram supports buttons, but Twitter and intelligent speakers do not.

3. Choosing a tool based on technical factors

The growing popularity of chatbots has caused the emergence of many tools for their construction. These range from low-level NLP services helping in the encoding of intents and their training phrases, to comprehensive low-code

development platforms covering most steps in the chatbot creation process.

Table 1 compares the main available software options for chatbot construction. It includes proposals of both large companies (*Dialogflow* by Google, *Watson* by IBM, *Lex* by Amazon, *Bot Framework* and *LUIS* by Microsoft) and younger chatbot specialized companies (*FlowXO*, *Landbot.io*, *Chatfuel*, *Rasa*, *SmartLoop*, *Xenioo*, *Botkit* which has been recently acquired by Microsoft, *ChatterBot* and *Pandorabots*). All are domain-independent but *Chatfuel*, which targets marketing applications.

The features analysed in the table stem from a thorough analysis of each tool. We distinguish between technical features (e.g., input processing) which are discussed in this section, and managerial features (e.g., pricing model) presented in the next section.

The first row in the table indicates whether the software is a library, a framework, a platform or a service. While platforms and frameworks offer support for the whole bot creation life-cycle, services and libraries support only some steps, typically related to NLP. Frameworks provide sets of classes that need to be complemented with custom code for each created chatbot, and hence chatbots are built via programming. Most platforms are cloud-based, low-code development environments to define chatbots graphically or via forms, and frequently support hosting the deployed chatbot logic for a channel. In addition, some platforms and frameworks (e.g., *Dialogflow*, *Bot Framework*, *Rasa*) also support the use of their NLP modules via services.

Rows 2–26 in the table analyse decisive technical dimensions when selecting a chatbot development tool. These comprise aspects related to the processing of the user input text (rows 2–7), the dialogue support (rows 8–13), the chatbot deployment (rows 14–15), the integration with other systems (rows 16–17), testing and development support (rows 18–22), execution support (rows 23–25) and security aspects (row 26).

Input processing. Some approaches allow defining the expected input phrases using regular expressions or patterns (row #2), while others permit specifying intents via training phrases and then apply NLP (row #3). In addition, platforms like *Landbot.io* also support user inputs by means of buttons and widgets. Most approaches based on NLP can identify parameters in the input phrases, with the exception of *Chatfuel* and *ChatterBot* (row #4). Another important aspect in NLP is the language support (row #5). All approaches consider some of the most spoken languages (English, Spanish), and some platforms excel for their wide language support (e.g., *Dialogflow* includes 22). Interestingly, *Rasa* can use pre-trained language models (e.g., *fastText* word vectors are available for hundreds of languages [9]) but developers can train their own. Only a few approaches – the NLP service *LUIS*, *Watson*, *Lex*, *Bot Framework,* and the Enterprise non-free edition of *Dialogflow* – provide sentence sentiment analysis, which can be useful in specific domains such as marketing. Finally, in addition to text, several approaches natively support voice-based interaction (row #7). This interaction kind could be added by hand to approaches based on programming languages (e.g., *Botkit*) or which are open source.

Dialogue. This dimension looks at the capabilities to organize the conversation flow. All platforms and most frameworks automatically store the parameter values extracted from user phrases to allow their reuse in the future, while libraries require programming this facility (row #8). This storage can be volatile (active only during the current user interaction) or persistent. Intents and entities (rows #9 and #10) are common primitives of platforms like *Dialogflow*, *Watson* and *Lex*. Approaches

Table 1: Comparison of chatbot libraries, frameworks, platforms and services.

			Dialogflow (Google) [v2]	Watson (IBM) [v2]	Lex (Amazon) [07/06/2020]	Bot Framework + LUIS (Microsoft) [v4]	FlowXO [07/06/2020]	Landbot.io [07/06/2020]	Chatfuel [07/06/2020]	Rasa [10.1.2]	SmartLoop [07/06/2020]	Xenioo [07/06/2020]	Botkit (part of Bot Framework) [4.9.0]	LUIS [05/192020]	ChatterBot [1.0.5]	Pandorabots [07/06/2020]
Technical Factors		1. Kind (**L**ibrary, **F**ramework, **P**latform, **S**ervice)	P	P	P	F	P	P	P	F	P	P	F	S	L	P
	Input processing	2. Regular expressions/patterns	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
		3. NLP for phrase match	✓	✓	✓	✓			✓	✓	✓	✓		✓	✓	
		4. Text processing to obtain phrase parameters	✓	✓	✓	✓				✓	✓	✓		✓		✓
		5. Number of languages: **v**ery high (≥50), **h**igh (≥10), **s**ome (<10), 1 (represented with flag)	h	h		h	h		h	v		s		h	v	
		6. Sentiment analysis	✓	✓	✓	✓								✓		
		7. Speech recognition	✓	✓	✓	✓						✓		✓		
	Dialogue	8. Storage of phrase parameters: **v**olatile, **p**ersistent, **b**oth	b	b	b	b	b	v	v	b	v	v		v		v
		9. Support for intents	✓	✓	✓	✓	✓			✓	✓			✓		✓
		10. Support for entities: **p**redefined, **u**ser-def, **b**oth	b	b	b	b	p	p		b	b	b		b		
		11. Dialogue structure: **t**ree, **f**ollowup intents, **d**sl	f	f	f	f	t	t	t	t	f	t			f	d
		12. Utterances to reengage users					✓		✓		✓	✓				
		13. Specification of chatbot answers	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓
	Deployment	14. Integration with social networks/websites: **h**igh (≥10), **s**ome (<10), 1 (represented with logo)	h	s	s	h	s			h	s	s	s			s
		15. Interaction support for specific social networks	✓			✓						✓				✓
	Sys. integ.	16. Call to services from chatbot	✓	✓	✓	✓	✓	✓	✓	✓					✓	
		17. Chatbot usage via API	✓	✓	✓						✓	✓		✓		✓
	Development and testing	18. Pre-built components: **c**hatbot templates, **i**ntents, small **t**alks, **s**ervices	cts	c	i	cs	c	c							c	t
		19. Version control: **n**ative, **c**ode-based	n	n	n	c				c			c		c	
		20. Chat console for testing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓
		21. Debug mechanisms	✓			✓				✓			✓		✓	
		22. Validation support	✓													
	Execution	23. Hosted deployment	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓		✓
		24. Support for analytics	✓	✓		✓	✓	✓			✓	✓				
		25. User message persistence	✓		✓	✓	✓	✓	✓			✓	✓			
	Sec	26. Cloud security	✓	✓	✓	✓								✓		
Managerial Factors	Organization	27. Pricing model: **f**ree, **p**ay-as-you-go, **q**uota, **a**dvanced feats	fp	fp	fp	fpa	fq	fa	fa	fa	fa	fq	f	fq	f	fqa
		28. Developer expertise: **l**ow, **h**igh	l	l	l	h	l	l	l	h	l	l	h	h	h	l
	Development	29. Code hosting: **e**xternal, **o**n-premises	e	e	e	o	e	e	e	o	e	e	o	o	o	e
		30. Group work				✓				✓			✓	✓	✓	
		31. i8n	✓													
		32. Open source								✓			✓		✓	
	Operational	33. New channels								✓			✓	✓	✓	
		34. No vendor lock-in								✓			✓	✓	✓	

supporting NLP define intents by sets of training phrases. These phrases may be examples of expected user utterances, or to improve the user experience, they may be obtained from existing conversation logs (e.g., when migrating a traditional customer support system into a chatbot). Regarding the dialogue structure (row #11), we find two main definition styles: explicitly by means of a conversation tree where nodes correspond to dialogue steps, or implicitly via dependent contexts and

follow-up intents which are activated upon matching their parent intent (e.g., an intent for making appointments which declares a follow-up intent to inform the kind of pet). More differently, *Pandorabots* uses the Artificial Intelligence Markup Language (AIML, http://www.aiml.foundation/), an XML format from the '90s aimed to be a scripting standard for chatbots. Being based on templates, it is in stark contraposition to modern approaches based on NLP. Some platforms also permit defining utterances that the chatbot can use to reengage unresponsive users (row #12). Finally, all approaches but *LUIS* and *Botkit* permit specifying the chatbot answers (row #13).

Deployment. While some approaches allow deploying chatbots in many social networks, others target specific ones (row #14). For example, *Chatfuel* chatbots are specific for Facebook messenger, *Landbot.io* chatbots can be deployed just on *WhatsApp Business* and websites, while *Dialogflow* has 15 channel integrations including websites, services like Skype, intelligent speakers and social networks like Slack, Viber, Twitter and Telegram. Libraries and services lack deployment options, since this is out of their scope. In addition, *Dialogflow*, *Bot Framework*, *Xenioo* and *Pandorabots* permit including custom interaction mechanisms for the selected channel, like buttons in Telegram (row #15).

System integration. Several approaches enable calling services from the chatbots (row #16). In some cases, like *Dialogflow*, this is done by associating the URL of the service to an intent, so that matching the intent triggers a POST message to the service. In other cases, it is possible to define programs with custom code. For this purpose, *Dialogflow* supports Cloud Functions for Firebase, and *Lex* supports AWS lambdas.

Conversely, some approaches offer an API that permits integrating parts of the chatbots with existing applications (row #17). For example, *Dialogflow* chatbots can be used programmatically to check the most probable matching intent given a user phrase.

Development and testing. Some tools offer pre-built components that can be added into new chatbots (row #18). These include generic chatbot templates (e.g., for a coffee shop or a hotel booking system), predefined intents, predefined small talks (answers to simple phrases and questions), or services (e.g., to build a *Q&A* chatbot from a knowledge base). Regarding version control (row #19), all frameworks and libraries rely on code and can be used with any generic version control system, while only some platforms (*Dialogflow*, *Watson* and *Lex*) give native support for versioning though this is simpler than state-of-the-art versioning systems like *GitHub*.

As for testing, most approaches provide a web chat console to test the chatbots manually (row #20). For debugging (row #21), frameworks and libraries can rely on the support of the programming language, while only one platform (*Dialogflow*) offers debugging facilities to inspect the matched intent and related information. In addition, *Dialogflow* incorporates checks of the chatbot quality, such as detecting intents with similar training phrases (row #22).

Execution. Once a chatbot is defined, all platforms and most frameworks support its execution on their clouds (row #23). This solution can be optimal for many companies, especially if they already use the cloud services of the platform vendor (e.g., Google, Azure or AWS); however, this may not be always suitable. In some cases, like *Watson*, there is a special pricing plan to deploy the chatbot on third-party clouds. Finally, some approaches permit obtaining analytics about the chatbot usage (row #24) or persisting the user phrases (row #25). Developers might find the latter feature useful to adjust the accuracy of the intent recognition and improve the user experience [10]. Approaches like *Watson* automate this

task, while others like *Dialogflow* require uploading the conversation logs and retraining.

Security. Chatbots may need to incorporate security aspects, especially if they work with private user data. While in general, implementing any security capability is the developers' responsibility, some tools can provide a security layer atop the cloud where the chatbot is deployed (row #26). Hence, approaches without deployment services do not offer this possibility natively. Instead, *Dialogflow*, *Watson*, *Lex* and Azure (Microsoft cloud for the *Bot Framework* and *LUIS*) provide a layer with features like firewalls; authentication and authorization when used via API; and secure connections (e.g., SSL or HTTPS/TLS). In addition, social networks like Whatsapp or Telegram support message encryption and user authentication.

4. Adding managerial factors to the equation

In addition to technical factors, some managerial factors may influence the selection of a development tool. Rows 27–34 in Table 1 classify those factors among organizational, related to development, or operational. We elicited those factors by a thorough analysis of the tools' features, and classified them using as a basis typical concerns in software projects.

Organizational factors. A critical selection factor is the pricing model of the approach (row #27). Most offer a free version suitable for small businesses or for experimentation (e.g., *Dialogflow* provides five free assistants and *Watson* supports 10,000 API calls). In addition, they provide other pricing models, typically collecting small fees for every interaction with the chatbot (the pay-as-you-go option of *Dialogflow*), limiting the number of interactions or active chatbots (the different plans of *FlowXO*), or supplying advanced features (e.g., webhooks in *Landbot.io* are not free).

The expertise of the development team on chatbot-related technology is also important (row #28). Development platforms allow creating simple chatbots with no need for coding and require less expertise than approaches based on programming, though these latter are less constrained.

Development related factors. Like any kind of software, chatbot construction should follow proper engineering processes. In this respect, using a platform may be problematic if the chatbot development has to be harmonized with the company development culture and processes. For example, platforms host the chatbot specifications on their clouds, while the backend needs to reside in a different place; instead, chatbots developed with libraries, frameworks and services can run on-premises (row #29). Likewise, some code facilities such as versioning or debugging are standard for frameworks and libraries but may be unavailable for some platforms. The same applies to group work (row #30): platforms currently do not support synchronous collaborative development, so working on different parts of a chatbot cannot be parallelised among developers.

Depending on the domain or the company strategy, the need to support several languages (i8n) can be necessary (row #31). Rather than developing a chatbot for each language, platforms like *Dialogflow* offer multi-language support by enabling the specification of different training phrases for each language over the same intent.

Interestingly, among the reviewed approaches, only the community edition of *Rasa*, *Botkit* and *ChatterBot* are open source (row #32). No platform is open source, which may result in vendor lock-in, but it is possible to make public the chatbot specifications built with any platform.

Operational factors. Once a chatbot is in operation, the need to deploy it in novel

channels or new versions of existing ones may arise (row #33). If the chatbot was developed using a platform, the available deployment options might be limited (e.g., *Watson* does not provide out-of-the-box deployment in Telegram). Libraries and (extensible) frameworks like *Rasa*, *Botkit*, *LUIS* and *ChatterBot* are more flexible, as they allow the manual implementation of the required deployment.

Finally, platform-based approaches imply vendor lock-in as there are currently no migration tools using neutral exchange formats between platforms (row #34); however, an advantage of platforms is the ability to use the services of the provider (IBM, Google). Instead, libraries and frameworks require coding the chatbot logic in a programming language (like Python in case of *Rasa*), which brings more independence and safety with respect to possible policy changes of the platform owner company. This independence is stronger in open-source systems (row #32) since they could even be personalized to the developer needs.

5. Building a chatbot in practice

Practitioners can exploit the information in Table 1 to select the best tool depending on the scenario. While this analysis can be hand-crafted, we envision a recommender system that automatically identifies the optimal tools from the chatbot requirements.

As an illustration, let's assume two scenarios for our vet clinic chatbot. In the first one, the clinic wants to reach as many potential clients as possible, so it asks for a chatbot that is multi-language and works on different social networks and intelligent speakers. Moreover, the software company that will develop the chatbot lacks the infrastructure to host the bot. Given these requirements, the only suitable chatbot creation tool is *Dialogflow*.

In the second scenario, the clinic is in a process of expansion so the chatbot may be likely extended in the future. Hence, the software company is thinking of using either *Rasa* or *Botkit* to avoid vendor lock-in. Since the company has an expert team of Python developers, and wants to have support for debugging and testing, it opts for *Rasa*.

We have built prototypical chatbots using the tools selected in the scenarios: *Dialogflow* and *Rasa*. The chatbots communicate with a backend that holds a database written in Java and PostgreSQL. The chatbots for Telegram, including their specification, are available at https://github.com/SaraPerezSoler/VetClinic.

The chatbot specification in *Dialogflow* has four intents: a welcome intent, a fall-back intent, an intent to query the opening hours, and another to set appointments. The welcome and fall-back intents were predefined in *Dialogflow* and reused in our chatbot without modification. To make the chatbot multi-language, each intent has to be trained with phrases in every targeted language. The appointment intent has a follow-up asking for the type of pet. This control flow is specified via a context. We defined an entity to recognise pet types, and reused the *date* and *time* system entities. The backend is accessed by a webhook that calls the database service via a POST request; alternatively, the behaviour could be implemented with a JavaScript in-line editor available in the platform. The deployment in Telegram was straightforward using *Dialogflow*'s integration options, and there are integrations for intelligent speakers as well.

Differently from *Dialogflow*, creating a chatbot in *Rasa* is not done via a graphical interface, but requires programming in Python and defining configuration files (YAML and markdown) storing the entities, intents, conversation flow, training phrases, bot responses, actions, forms, NLP configuration and credentials to access external services. The *Rasa* chatbot has one fall-back action and three intents:

greeting, *time* and *make_appointment*. To define the parameters of the last intent, we subclassed a specific *Rasa* class to store the name and type of the parameters, validation methods, and other details. The chatbot actions (e.g., querying the database, calling external services) were programmed in Python as well. The chatbot behaviour can be debugged and tested using standard Python tooling. Unlike *Dialogflow*, the developer must perform the chatbot deployment as *Rasa* does not host bots.

6. Open Challenges

Overall, the existing tools cover a wide spectrum of possibilities to ease chatbot creation in different scenarios. However, designing, developing and testing chatbots still pose some challenges. First, most platforms offer general, informal guidelines for chatbot design, but design patterns and quality metrics for chatbots are missing. With regards to development, most tools rely on training phrases to specify intents; while this is suitable in closed domains, supporting less constrained conversations would require the tools to incorporate more sophisticated NLP mechanisms [13, 14] and better support to expand the training set using techniques such as reinforcement learning (e.g., via trial-and-error conversations with real or simulated users). Also related to quality, existing tools give poor support for testing chatbots in a systematic and automated manner; at best, they provide a console for manual testing, and basic debugging mechanisms (rows 20–21 in Table 1). Some dedicated testing tools are emerging, like https://www.botium.at/.

Ultimately, the success of a chatbot depends on its usability and the user experience. Some technical factors in Table 1 may help to improve this usability: NLP enables more natural conversations, phrase parameters avoid users to provide a different sentence per piece of information, sentiment analysis can contribute to better grasp the meaning of a phrase and act accordingly, speech recognition supports spoken conversation, rich dialog structuring mechanisms allow more sophisticated conversation flows, and message persistence can be exploited to improve chatbot accuracy by the analysis of real conversations. To complement this, chatbot development tools should invest in embedding guidelines and heuristics targeted to chatbot usability [11, 12].

Chatbot development tools are rapidly expanding, but we believe that after diversification comes unification. The analysed technologies use their own proprietary formats to define chatbots, and automated migration tools are missing. To unify the different approaches, the W3C is developing a standard for conversational agents (https://www.w3.org/community/conv/), and some open-source initiatives aim to integrate the best of every chatbot platform, helping to solve the vendor lock-in problem [15].

References

1. PAT Research. "*How to select the best chatbot platforms for your business?*". Available at https://www.predictiveanalyticstoday.com/what-is-chatbot-platform/ (last visited Jan-2020).
2. OMetrics. "*2019 chatbot platform comparison reviews*". Available at: https://www.ometrics.com/blog/chatbot-platform-comparison-reviews/ (last visited Jan-2020).
3. Davydova. "*25 chatbot platforms: A comparative table*". Chatbots Journal (May 2017). Available at: https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff (last visited Jan-2020).
4. VentureHarbour. "*10 best chatbot builders in 2019*". https://www.ventureharbour.com/best-chatbot-builders/ (last visited Jan-2020).
5. Lebeuf, Storey, Zagalasky. "*Software bots*". IEEE Software 35(1): 18–23 (2018).
6. Marneffe, Maccartney, Manning. "*Generating typed dependency parses from phrase structure parses*". Proc. LREC'06: 449–454. See also https://nlp.stanford.edu/software/lex-parser.html (last visited Jan-2020).
7. Bird, Klein, Loper. "*Natural language processing with Python - Analyzing text with the Natural*

Language Toolkit". O'Reilly 2009. See also https://www.nltk.org/ (last visited Jan-2020).

8. Pereira, Díaz. "*Chatbot dimensions that matter: Lessons from the trenches*". Proc. ICWE'18: 129–135.
9. Grave, Bojanowski, Gupta, Joulin, Mikolov. "*Learning word vectors for 157 languages*". Proc. LREC 2018.
10. Hancock, Bordes, Mazaré, Weston. "*Learning from dialogue after deployment: Feed yourself, chatbot!*". Proc. ACL (1) 2019: 3667–3684.
11. Ren, Castro, Acuña, de Lara. "*Usability of chatbots: A systematic mapping study*". Proc. SEKE'19: 479–617.
12. Haptiki.ai. "*10 usability heuristics to design better chatbots*". https://haptik.ai/blog/usability-heuristics-chatbots/ (last visited Jan-2020).
13. Pérez-Soler, Guerra, de Lara, Jurado. "*The rise of the (modelling) bots: Towards assisted modelling via social networks*". Proc. ASE'17: 723–728.
14. Arora, Sabetzadeh, Nejati, Briand. "*An active learning approach for improving the accuracy of automated domain model extraction*". ACM Trans. Softw. Eng. Methodol. 28(1): 4:1–4:34 (2019).
15. Daniel, Cabot, Deruelle, Derras. "*Multi-platform chatbot modeling and deployment with the Jarvis framework*". Proc CAiSE'19: 177–193. See also https://xatkit.com/ (last visited Jan-2020).