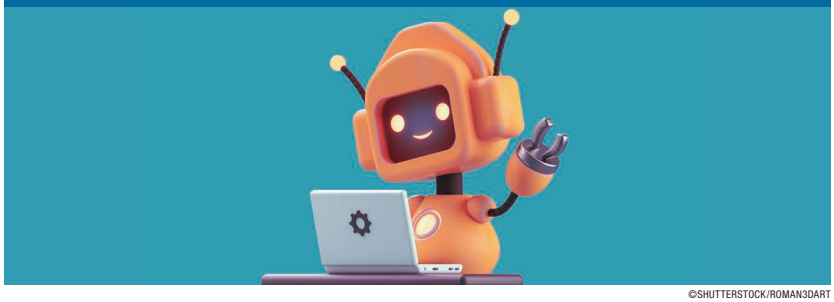
Choosing a Chatbot Development Tool

Sara Pérez-Soler, Sandra Juárez-Puerta, Esther Guerra, and Juan de Lara, Universidad Autónoma de Madrid

// Chatbots are programs that supply services to users via conversation in natural language, acting as virtual assistants within social networks or web applications. Here, we review the most representative chatbot development tools with a focus on technical and managerial aspects. //



©SHUTTERSTOCK/ROMAN3DART

CHATBOTS ARE PROGRAMS with a conversational user interface. Their popularity is rising because they enable access to all sorts of services (for example, booking flights and 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).[1]

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, and 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 noncompliance with chatbot technical requirements or with software development company policies. Some websites and informal blogs compare some available options to build chatbots,[2–5] and researchers have identified aspects to consider in chatbot design (functional, integration, analytics, and quality assurance).[6] Instead, we analyze 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 (for example, "we need a chatbot to access our current information system by text and voice, in both English and Spanish") and managerial constraints (for example, "my developers lack experience in developing chatbots, and we do not have the capacity to deploy on-premises, and we are already using Amazon cloud").

What's in a Chatbot?

A chatbot is a program supporting user interaction via conversation in NL, and it is normally accessible through the web or social networks. As an example, assume that a

Digital Object Identifier 10.1109/MS.2020.3030198
Date of current version: 18 June 2021

0740-7459/21©2021IEEE

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 that customers can ask about opening hours and make appointments. This chatbot would allow the clinic to offer 24/7 service, reduce costs (for example, decreasing customer telephone calls), and widen the range of potential customers. Figure 1(a) depicts an example of a user interacting with the envisioned chatbot.

As Figure 1(b) shows, a chatbot is organized around intents that represent the possible user intentions and permits that person to access the offered services. These intents typically reflect the 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 Figure 1(b)], the chatbot identifies the matching intent (label 2). Depending on the intent, the chatbot may need to access the external services, like the clinic database, if the intent is setting an appointment (label 3). Finally, the chatbot replies to the user, for example, confirming the appointment (label 4).

Figure 2(a) illustrates a process diagram with the main activities that designing a chatbot entails. The development process is not necessarily linear; it 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[7] or the Natural Language Toolkit[8]—as they permit analyzing the phrase structure, and they 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 example vet clinic), it is simpler to train the chatbot with training phrases (that is, examples of expected phrases) characterizing each intent. We can find libraries and services that apply machine learning for this purpose, like Microsoft's Language Understanding (LUIS) (https://luis.ai) or Rasa Natural Language Understanding (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

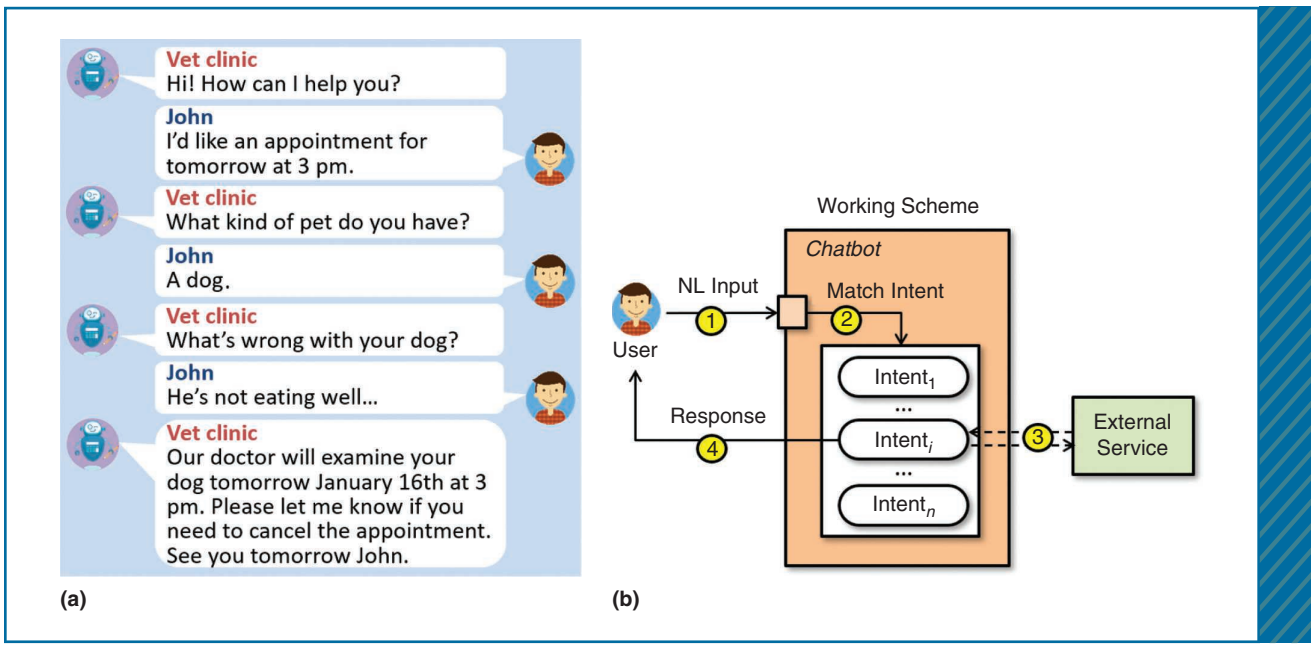


FIGURE 1. (a) An example of user interaction. (b) The working scheme of a chatbot.

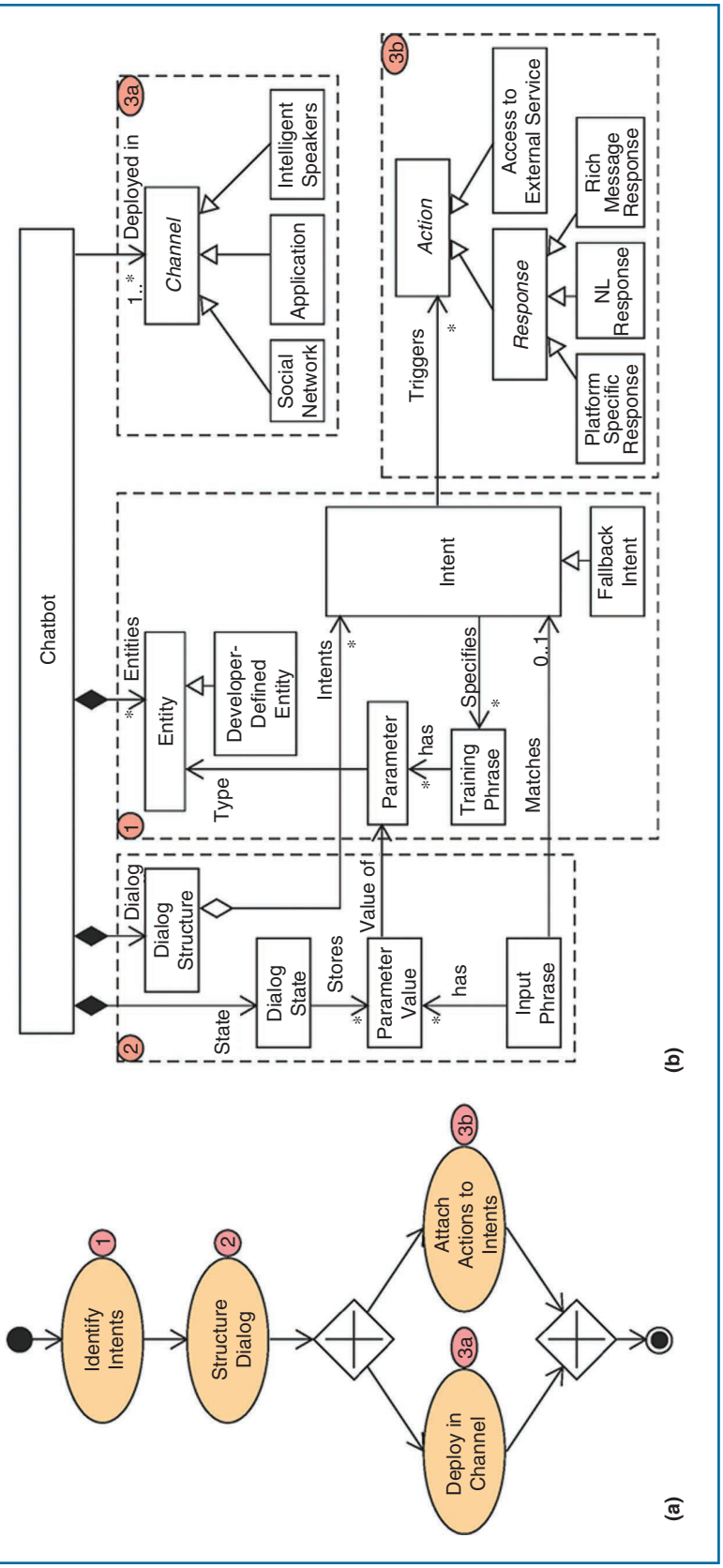


FIGURE 2. (a) A process diagram for chatbot design. (b) A structural diagram of chatbot concepts.

entity type. Most chatbot development tools provide predefined entities (for example, dates and numbers), and developers can define new ones (for example, pet types). In addition, chatbots may define fallback intents, used when the chatbot does not recognize the user utterance.

Besides intents, developers need to define the dialog 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 is the appointment fixed. For this purpose, the chatbot needs to store the dialog 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 on some channel. Typical channels are social networks, websites, or smart speakers like the 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 (such as 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.

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, and 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 except for Chatfuel, which targets marketing applications.

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

The first row in Table 1 indicates whether the software is a library, framework, platform, or service. While platforms and frameworks offer support for the whole bot creation lifecycle, 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 they frequently support hosting the deployed chatbot logic for a channel. In addition, some platforms and frameworks (for example, Dialogflow, Bot Framework, and Rasa) also support the use of their NLP modules via services.

Rows 2–26 in Table 1 analyze 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 dialog support (rows 8–13), the chatbot deployment (rows 14–15), 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 of NLP is language support (row 5). All approaches consider some of the most-spoken languages (such as English and Spanish), and some platforms excel for their wide language support (for example, Dialogflow includes 22 languages). Interestingly, Rasa can use pretrained language models (for example, fastText word vectors are available for hundreds of languages)[9], but developers can train their own. Only a few approaches—the NLP services LUIS, Watson, Lex, Bot Framework, and the Enterprise nonfree 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 (for example, Botkit) or which are open source.

Dialog

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 (that is, 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 supporting NLP define intents by sets of training phrases. These phrases may be examples of expected user utterances or phrases to improve the user experience, and they may be obtained from existing conversation logs (for example, when migrating a traditional customer support system into a chatbot).

Regarding the dialog structure (row 11), we find two main definition styles: explicitly by means of a conversation tree where nodes correspond to dialog steps or implicitly via dependent contexts and follow-up intents that are activated upon matching their parent intent (for example, an intent for making appointments that declares a follow-up intent to inform the kind of pet). More differently, Pandorabots uses the Artificial Intelligence Markup Language (http://www.aiml.foundation/), an XML format

Table 1. A comparison of chatbot libraries, frameworks, platforms, and services.

Technical factors		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 [05/19/2020]	ChatterBot [1.0.5]	Pandorabots [07/06/2020]
		P	P	P	F	P	P	P	F	P	P	F	S	L	P
Input processing	1. Kind (Library, Framework, Platform, Service)	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
	2. Regular expressions/patterns	✓	✓	✓	✓			✓	✓	✓	✓		✓	✓	
	3. NLP for phrase match	✓	✓	✓	✓			✓	✓	✓	✓		✓		✓
	4. Text processing to obtain phrase parameters	✓	✓	✓	✓				✓	✓	✓		✓		
	5. Number of languages: very high (≥50), high (≥10), some (<10), one (represented by U.S. flag)	h	h		h	h		h	v		s		h	v	
	6. Sentiment analysis	✓	✓	✓	✓								✓		
	7. Speech recognition	✓	✓	✓	✓						✓		✓		
Dialog	8. Storage of phrase parameters: volatile, persistent, both	b	b	b	b	b	v	v	b	v	v		v		v
	9. Support for intents	✓	✓	✓	✓	✓			✓	✓			✓		✓
	10. Support for entities: predefined, user-defined, both	b	b	b	b	p	p		b	b	b		b		
	11. Dialog structure: tree, follow-up intents, dsl	f	f	f	f	t	t	t	t	f	t			f	d
	12. Utterances to re-engage users					✓		✓			✓				
	13. Specification of chatbot answers	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓
Deployment	14. Integration with social networks/websites: high (≥10), some (<10), one (represented with logo)	h	s	s	h	s			h	s	s	s			s
	15. Interaction support for specific social networks	✓			✓						✓				✓
System integration	16. Call to services from chatbot	✓	✓	✓	✓	✓	✓	✓	✓					✓	
	17. Chatbot usage via API	✓	✓	✓						✓	✓		✓		✓

(*Continued*)

Table 1. A comparison of chatbot libraries, frameworks, platforms, and services. (cont.)

		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 [05/19/2020]	ChatterBot [1.0.5]	Pandorabots [07/06/2020]
Technical factors	Development and testing	cts	c	i	cs	c	c							c	t
	18. Prebuilt components: chatbot templates, intents, small talks, services	n	n	n	c				c			c		c	
	19. Version control: native, code based	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓
	20. Chat console for testing	✓			✓				✓			✓		✓	
	21. Debug mechanisms	✓													
	22. Validation support		✓	✓	✓	✓	✓	✓		✓	✓		✓		
	Execution	✓	✓		✓	✓	✓			✓	✓				
	23. Hosted deployment	✓	✓		✓	✓	✓				✓				
	24. Support for analytics	✓		✓	✓	✓	✓	✓			✓	✓			
	25. User message persistence	✓	✓		✓										
	Security	✓	✓		✓								✓		
	26. Cloud security														
Managerial factors	Organization	fp	fp	fp	fpa	fq	fa	fa	fa	fa	fq	f	fq	f	fqa
	27. Pricing model: free, pay-as-you-go, quota, advanced feats	l	l	l	h	l	l	l	h	l	l	h	h	h	l
	28. Developer expertise: low, high	e	e	e	o	e	e	e	o	e	e	o	o	o	e
	Development				✓				✓			✓	✓	✓	
	29. Code hosting: external, on-premises	✓													
	30. Group work														
	31. i18n									✓		✓		✓	
	32. Open source								✓						
	Operational								✓			✓	✓	✓	
	33. New channels														
	34. No vendor lock-in								✓			✓	✓	✓	

v: version.

from the 1990s that aimed at being 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 re-engage 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, and Landbot.io chatbots can be deployed only 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 the inclusion of 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 Amazon Web Services (AWS) Lambda.

Conversely, some approaches offer an application programming interface (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 prebuilt components that can be added into new chatbots (row 18). These include generic chatbot templates (for example, for a coffee shop or a hotel booking system), predefined intents, predefined small talks (for example, answers to simple phrases and questions), or services (for example, to build a question and answer 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 (for example, Dialogflow, Watson, and Lex) give native support for versioning, although 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 [for example, Google, Azure (Microsoft's cloud for the Bot Framework and LUIS), or AWS]; however, this may not always be 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 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 provide a layer with features like firewalls; authentication and authorization when used via API; and secure connections (for example, Secure Sockets Layer or HTTPS/Transport Layer Security). In addition, social networks like WhatsApp or Telegram support message encryption and user authentication.

Adding Managerial Factors to the Equation

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

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 (for example, 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 (such as the pay-as-you-go option of Dialogflow), limiting the number of interactions or active chatbots (for example, the different plans of FlowXO), or supplying advanced features (for example, the 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 for 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's 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 the 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 parallelized among developers.

Depending on the domain or the company strategy, the need to support several languages (i18n) can be necessary (row 31). Rather than developing a chatbot for each language, platforms like Dialogflow offer multilanguage support by enabling the specification of different training phrases for each language over the same intent.

Interestingly, among the reviewed approaches, only the community editions 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 (for example, 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 (for example, IBM and Google). Instead, libraries and frameworks require coding the chatbot logic in a programming language (like Python in the 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 developers' needs.

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 handcrafted, 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 multilanguage and works on different social networks and with 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 the 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 fallback intent, an intent to

query the opening hours, and another to set appointments. The welcome and fallback intents were predefined in Dialogflow and reused in our chatbot without modification. To make the chatbot multilanguage, each intent had 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 recognize 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 behavior can 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 was not done via a graphical interface but required 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 fallback 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 (for example, querying the database and calling external services) were programmed in Python as well. The chatbot behavior can be debugged and tested using standard Python tooling. Unlike Dialogflow, the developer must perform the chatbot deployment as Rasa does not host bots.

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 regard 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[11,12] and better support to expand the training set using techniques such as reinforcement learning (for example, 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 Botium (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 having 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.[13,14]

Chatbot development tools are rapidly expanding, but we believe that after diversification comes unification. The analyzed technologies use their own proprietary formats to define chatbots, and automated migration tools are missing. To unify the different approaches, the World Wide Web Consortium 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]

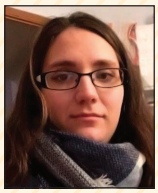
Acknowledgments

This work was partially funded by the R&D program of the Madrid Region (project FORTE, S2018/TCS-4314), and the Spanish Ministry of Science (project MASSIVE, RTI2018-095255-B-I00).

References

1. C. Lebeuf, M. Storey, and A. Zagalasky, "Software bots," *IEEE Softw.*, vol. 35, no. 1, pp. 18–23, 2018. doi: 10.1109/MS.2017.4541027.
2. "2019 chatbot platform comparison reviews." OMetrics. https://www.ometrics.com/blog/chatbot-platform-comparison-reviews/ (accessed Jan. 2020).
3. O. Davydova, "25 chatbot platforms: A comparative table." Chatbots Journal. https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff (accessed Jan. 2020).
4. A. Brooks, "10 best chatbot builders in 2019." VentureHarbour. https://www.ventureharbour.com/best-chatbot-builders/ (accessed Jan. 2020).

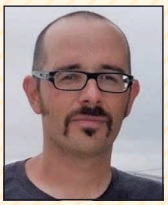
ABOUT THE AUTHORS



SARA PÉREZ-SOLER is a Ph.D. candidate in computer science and an assistant professor at the Universidad Autónoma de Madrid, Madrid, 28049, Spain. Her research interests include conversational agents, model-driven engineering, and automated software development. Pérez-Soler received her M.Sc. in computer science from the Universidad Autónoma de Madrid. Contact her at sara.perezs@uam.es.

SANDRA JUÁREZ-PUERTA is a member of the Modelling & Software Engineering Research Group at the Universidad Autónoma de Madrid, Madrid, 28049, Spain. Her research interests include conversational agents, model-driven engineering, and automated chatbot development. Juárez-Puerta received her M.Sc. in computer science from the Universidad Autónoma de Madrid. Contact her at sandra.juarez@uam.es.

ESTHER GUERRA is an associate professor at the Universidad Autónoma de Madrid, Madrid, 28049, Spain. Her research interests include model-driven engineering, automated software development, and domain-specific languages. Guerra received her Ph.D. in computer science from the Universidad Autónoma de Madrid. Further information about her can be found at http://www.eps.uam.es/~eguerra/. Contact her at esther.guerra@uam.es.



JUAN DE LARA is a full professor at the Universidad Autónoma de Madrid, Madrid, 28049, Spain. His research interests include model-driven engineering, including metamodeling, model transformations, and domain-specific languages. de Lara received his Ph.D. in computer science from the Universidad Autónoma de Madrid. Further information about him can be found at http://www.eps.uam.es/~jlara/. Contact him at juan.delara@uam.es.

5. "How to select the best chatbot platforms for your business?" Predictive Analytics Today. https://www.predictiveanalyticstoday.com/what-is-chatbot-platform/ (accessed Jan. 2020).
6. D. Pereira, "Chatbot dimensions that matter: Lessons from the trenches," in *Proc. ICWE*, 2018, vol. 18, pp. 129–135.
7. M-C de Marneffe, B. Maccartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," in *Proc. LREC*, vol. 6, 2020, pp. 449–454.
8. S. Bird, E. Klein, and E. Loper, *Natural Language Processing With Python—Analyzing Text With the Natural Language Toolkit*. Cambridge: O'Reilly, 2009. [Online]. Available: https://www.nltk.org/ (accessed Jan 2020).
9. E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proc. LREC*, 2018.
10. B. Hancock, A. Bordes, P-E. Mazaré, and J. Weston, "Learning from dialogue after deployment: Feed yourself, chatbot!" in *Proc. ACL*, 2019, pp. 3667–3684. doi: 10.18653/v1/P19-1358.
11. S. Pérez-Soler, E. Guerra, J. de Lara, and F. Jurado, "The rise of the (modelling) bots: Towards assisted modelling via social networks," in *Proc. 2017 32nd IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, pp. 723–728. doi: 10.1109/ASE.2017.8115683.
12. C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, "An active learning approach for improving the accuracy of automated domain model extraction," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 1, pp. 4:1–4:34, 2019. doi: 10.1145/3293454.
13. R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara, "Usability of chatbots: A systematic mapping study," in *Proc. SEKE*, 2019, vol. 19, pp. 479–617. doi: 10.18293/SEKE2019-029.
14. "10 usability heuristics to design better chatbots." Haptiki. https://haptik.ai/blog/usability-heuristics-chatbots/ (accessed Jan. 2020).
15. G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Multi-platform chatbot modeling and deployment with the Jarvis framework," in *Proc CAiSE*, vol. 19, 2020, pp. 177–193.