Creating and Migrating Chatbots with CONGA

Sara Pérez-Soler
Universidad Autónoma de Madrid
Madrid, Spain
sara.perezs@uam.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

Abstract—**Chatbots are agents that enable the interaction of users and software by means of written or spoken natural language conversation. Their use is growing, and many companies are starting to offer their services via chatbots, e.g., for booking, shopping or customer support. For this reason, many chatbot development tools have emerged, which makes choosing the most appropriate tool difficult. Moreover, there is hardly any support for migrating chatbots between tools.**

To alleviate these issues, we propose a model-driven engineering solution that includes: (i) a domain-specific language to model chatbots independently of the development tool; (ii) a recommender that suggests the most suitable development tool for the given chatbot requirements and model; (iii) code generators that synthesize the chatbot code for the selected tool; and (iv) parsers to extract chatbot models out of existing chatbot implementations. Our solution is supported by a web IDE called CONGA that can be used for both chatbot creation and migration. A demo video is available at https://youtu.be/3sw1FDdZ7XY.

Index Terms—**Chatbots, Model-Driven Engineering, Domain-Specific Languages, Migration.**

I. INTRODUCTION

Chatbots are conversational agents that support interaction via natural language (NL) [1]. The improvements in NL processing have triggered their proliferation to access all kind of services, like flight booking, food delivery or customer support. By 2022, Gartner predicts that 70% of all customer interactions will involve machine learning, chatbots and mobile messaging[1]. Many companies are offering their services via chatbots to make them more accessible and user-friendlier, since chatbots are used via NL and can be deployed in social networks (called *channels*) like Telegram or Slack with no need to install dedicated apps [2].

Many chatbot development tools have emerged in recent years. Prominent software companies like Google, IBM, Microsoft or Amazon have launched products for chatbot development (Dialogflow[2], Watson[3], Microsoft Bot Framework[4], Amazon Lex[5]), but a plethora of other options exist, like Rasa[6], Xenioo[7] or Landbot.io[8], to name a few. This variety of tools poses several challenges to chatbot developers:

- *Challenge 1: How to identify the most appropriate development tool based on the chatbot requirements?* [3]. For example, only some tools offer off-the-shelf speech recognition, and tools wildly vary on the supported deployment channels. Choosing an inadequate tool may lead to increased effort [4], lower chatbot quality or project failure.
- *Challenge 2: How to design chatbots independently of the particular tool to enable early reasoning and analysis, prior to the implementation?* Chatbot development tools are very diverse, ranging from low-level programming frameworks (like Rasa) to lowcode development platforms based on forms (like Dialogflow). Grasping the design behind a chatbot implementation may be challenging due to accidental, technical details of the tools themselves.
- *Challenge 3: How to keep up with the rapidly evolving ecosystem of chatbot tools?* With a few exceptions [5], most chatbot tools are closed, proprietary software with no support for migration between tools, e.g., to benefit from the pricing plans of a competitor. This leads to vendor lock-in.

To address these challenges, we propose a web IDE called CONGA that offers a neutral domain-specific language (DSL) for chatbot modelling [6]. Chatbot models can be statically analysed to detect errors and quality issues, and be compiled into tools such as Rasa or Dialogflow. CONGA includes a recommender of suitable development tools for a given chatbot design. The recommender relies on the criteria identified in [3], and takes into account the chatbot model and the answers to a questionnaire of chatbot technical aspects (e.g., *is hosted deployment required?*) and managerial requirements (e.g., *pricing model*). Chatbot migration is facilitated by parsers from development tools into CONGA models, which in turn can be compiled into other platforms. The envisioned users of CONGA are developers and designers with conceptual knowledge on chatbots but not necessarily on their technologies.

This paper showcases the CONGA web IDE, which comprises a textual editor for chatbot modelling, graphical views of the designed conversation flow, a chatbot tool recommender, and generators/parsers to/from some prominent chatbot tools.

II. APPROACH

Next, we overview our approach (Section II-A) and describe its two main components: the DSL for chatbot modelling (Section II-B) and the recommender system (Section II-C).

A. Overview of the usage methodology of CONGA

We address the 3 challenges identified in the introduction by means of an automated process supporting both forward (i.e., creating new chatbots) and backward engineering (i.e., migrating existing chatbots). Fig. 1 depicts this process.

[1] https://www.gartner.com/smarterwithgartner/top-cx-trends-for-cios-to-watch/
[2] https://dialogflow.com/ [3] https://www.ibm.com/cloud/watson-assistant/
[4] https://dev.botframework.com/ [5] https://aws.amazon.com/en/lex/
[6] https://rasa.com/ [7] https://www.xenioo.com/en/ [8] https://landbot.io/

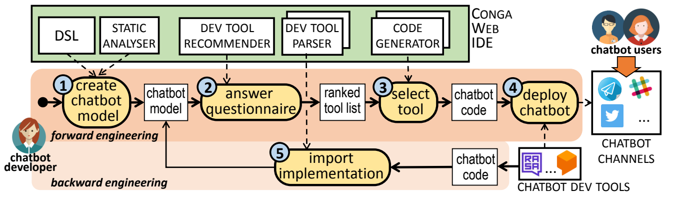


Fig. 1: Forward/backward chatbot engineering with CONGA.

Forward engineering. First, the developer describes the chatbot with a dedicated DSL (label 1 in the figure), explained in Section II-B. The result is a chatbot model that is independent of any development tool and can be statically analysed to detect flaws. Next, to get recommendations on suitable chatbot development tools, the developer answers a questionnaire on additional bot requirements beyond its functional behaviour (label 2). The recommender – detailed in Section II-C – analyses the developer's answers and the chatbot model to elaborate a ranked list of tools. This recommendation step is optional. Then, the developer selects a particular tool (label 3), and the system generates a fully functional chatbot implementation for it. Finally, the developer can deploy the chatbot on a channel (e.g., Telegram) using the selected tool (label 4).

Backward engineering. To support migration, the developer can import an existing chatbot implemented in a specific development tool, and CONGA parses the code to produce the corresponding chatbot model (label 5). The developer can then use this model for forward engineering.

B. The neutral DSL for chatbot modelling

CONGA provides a textual DSL for chatbot modelling, designed based on an analysis of 15 prominent chatbot development tools [6]. Listing 1 illustrates its usage to model a chatbot to help booking flights. First, line 1 declares the *languages* the chatbot should converse in, in this case just English (en), but multi-language chatbots are also possible.

Chatbots are designed around *intents*. These are the actions that users can perform with the bot, like booking or changing a flight. In CONGA, intents can be defined either by regular expressions, or by a set of training phrases showcasing typical ways in which users may express the intention (lines 5–10 in Listing 1). Training phrases may contain *parameters*, which are relevant data that the chatbot needs, such as the source, destination and date of a flight ("from", "to" and "when" in lines 6–7). Each parameter is formally declared by providing its type, whether it is optional or required, and in the latter case, a phrase that the chatbot should ask to the user to request a value for the parameter if it is missing (lines 11–14).

Parameters are typed by *entities* (lines 16–21), which can be pre-defined (like "date") or user-defined (like "City"). User-defined entities specify a set of entries and their synonyms.

Upon recognizing an intent, the chatbot can perform different *actions* such as replying to the user or accessing an external database. This is configured in an "actions" section (lines 23–35). The listing declares a text response (lines 24–27), an image response (lines 28–29) and a POST HTTP request

```
 1  chatbot FlightBooking language: en
 2
 3  intents:
 4    Book_flight:
 5      inputs {
 6        "I need to fly from" ("Madrid")[from] "to" ("Paris")[to]
 7          "on" ("Monday at 9 AM")[when],
 8        "I want to book a flight",
 9        "I need a flight to" ("Rome")[to]
10      }
11      parameters:
12        from: entity City, required, prompts ["What's the flight origin?"];
13        to: entity City, required, prompts ["What is the destination?"];
14        when: entity date, required, prompts ["When do you want to fly?"];
15
16  entities:
17    Simple entity "City":
18      inputs in en {
19        Madrid synonyms MAD, madrid
20        Rome synonyms ROM
21    }
22
23  actions:
24    text response fly_response:
25      inputs in en {
26        "Your flight from" [Book_flight.from] "to" [Book_flight.to] "is booked"
27    }
28    image response send_image:
29      URL: "https://image.shutterstock.com/image−vector.jpg"
30    Request post airline_service:
31      URL: "myURL.com";
32      basicAuth: "user":"pass";
33      headers: "header1":"value1";
34      data: "from": [Book_flight.from], "to": [Book_flight.to];
35      dataType: JSON;
36
37  flows:
38    − user Book_flight => chatbot airline_service, fly_response;
```

Listing 1: A chatbot for booking flights with CONGA (excerpt).

(lines 30–35). The text and the HTTP request use parameters gathered in the intent (Book_flight.from and Book_flight.to).

A last "flows" section permits defining conversation *flows* (lines 37–38). These are sequences of user intents (Book_flight) followed by chatbot actions (airline_service and fly_response). Flows can have any length, and there may be several possible user continuations after a chatbot action.

C. The recommender of chatbot development tools

CONGA models are not executable, but they can be compiled into code for a particular development tool. In previous work [3], we identified technical and managerial requirements influencing the tool selection process. To help in selecting an appropriate tool, CONGA integrates a recommender.

The recommender infers some tool requirements from the chatbot model, like the need to support multiple languages, user-defined entities or phrase parameters, among others. Additionally, the developer is presented a questionnaire concerning other non-functional requirements that may influence the tool selection but cannot be derived from the chatbot model. Some examples include the channels where the chatbot is to be deployed, support for chatbot analytics, speech recognition, or being open-source. Overall, the questionnaire has 10 questions [6], each with a customizable relevance that reflects its importance on the recommendation. This way, developers can specify that the answer to a given question is *irrelevant* (disregarded in the recommendation but stored for documentation), *relevant*, *double relevant* or *critical* (tools that do not fulfil the requirement will not be recommended).

The recommender uses the chatbot model and the answers to the questionnaire to assign a score to each development tool, where higher scores indicate wider requirements coverage.

III. Tool Support

This section describes the architecture (Section III-A) and front-end (Section III-B) of CONGA. The tool is open source, and is available at https://saraperezsoler.github.io/CONGA/.

A. Architecture

CONGA is available to chatbot developers as a web application. Fig. 2 shows its architecture. The front-end includes user and project managers, a DSL editor, a graphical renderer of conversation flow models, importers/exporters for some chatbot tools, a questionnaire for the tool recommender, and a visualizer of tool recommendations. The back-end handles the requests of the front-end concerning chatbot model validation, code generation, parsing, and recommendation computation.

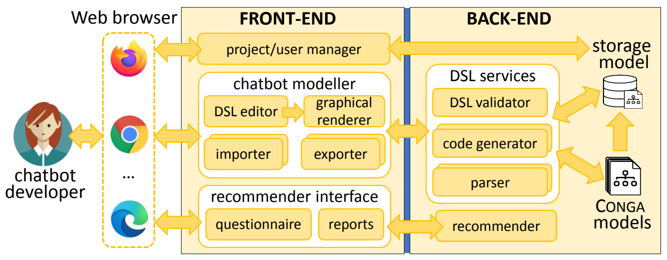


Fig. 2: CONGA's architecture.

The storage model of CONGA conforms to the meta-model of Fig. 3. It includes a *Recommender* class that defines the list of *Tools* that may be recommended (e.g., Dialogflow, Lex), and the *Requirements* considered to calculate the recommendation. There are two types of requirements: *Question*, which corresponds to a query in the recommender questionnaire (e.g., the deployment channels), and *Analysis*, which refers to technical requirements extracted from the chatbot model (e.g., the bot spoken languages). Both requirement types have a *name*, a *text* question, a closed list of response *options*, and can optionally be *multi*-option. Each tool considered for recommendation must define which of the specified requirements options are *available*, *unavailable*, *unknown* or might be *possible* in the tool. Currently, our recommender considers 10 questions, 7 model analyses, and 14 up-to-date target implementation tools; however, our model-based design makes the recommender fully extensible with new questions, analyses and tools.

Chatbot definitions are stored in *Projects*. Each project stores the developer's *Answers* to the *Questions* in the recommender *Questionnaire*. The answers comprise both the selected options and the relevance level assigned to the question.

B. Front-end

Fig. 4 shows the main interface of CONGA. The header (label 1) includes the logged user name, and a sign out button. The toolbar (label 2) contains buttons to save the file with the chatbot model, create a new project, format the displayed

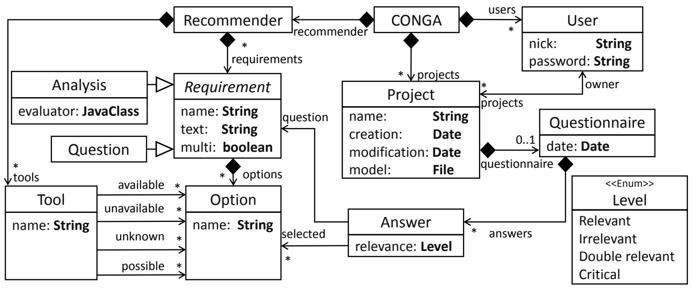
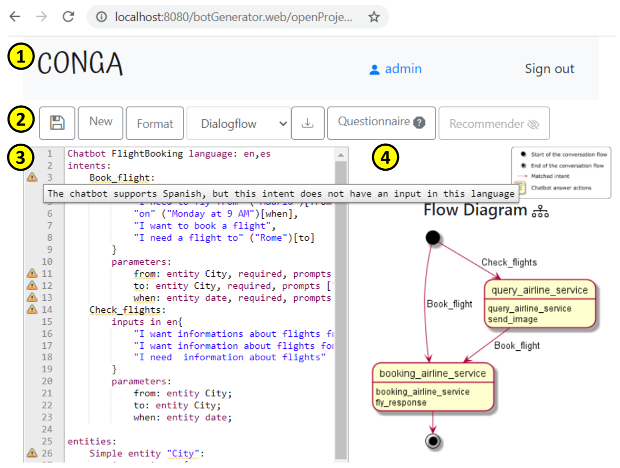


Fig. 3: CONGA storage meta-model.

file, select a development tool to generate code for (currently Dialogflow or Rasa), fill in the recommender questionnaire, and display the recommendation results. New projects can be created empty, or be populated with a model parsed from an existing chatbot implementation (currently from Dialogflow).



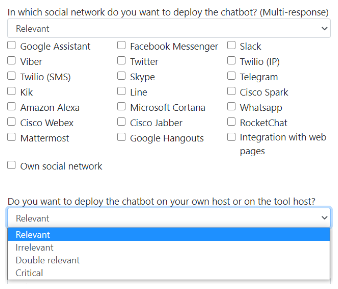
```
Chatbot FlightBooking language: en,es
intents:
    Book_flight:
The chatbot supports Spanish, but this intent does not have an input in this language
            "on" ("Monday at 9 AM")[when],
            "I want to book a flight",
            "I need a flight to" ("Rome")[to]
        }
        parameters:
            from: entity City, required, prompts
            to: entity City, required, prompts [
            when: entity date, required, prompts
    Check_flights:
        inputs in en{
            "I want informations about flights f
            "I want information about flights fo
            "I need  information about flights"
        }
        parameters:
            from: entity City;
            to: entity City;
            when: entity date;

entities:
    Simple entity "City":
```

Fig. 4: CONGA's main interface.

The DSL editor (label 3) features syntax highlighting, content assistance and error reporting. In addition to syntax errors, a validator checks problems like intents with overlapping training phrases, similar conversation flows, or flows where an action uses parameters that no previous intent in the flow defines. In the figure, the editor reports some warnings; the first one warns that the chatbot is multi-language (English, Spanish) but the training phrases only consider one language (English). Technically, the editor is implemented in Xtext, using its web deployment options for the codemirror JavaScript library.

The flow diagram to the right (label 4) depicts graphically the conversation flow defined by the chatbot model. The diagram represents the user interactions as transitions, and the chatbot interactions as states with the actions that the chatbot performs inside. This view is built using PlantUML, and becomes updated whenever the chatbot file is saved.

Fig. 5a shows an excerpt of the questionnaire that developers can answer to obtain tool recommendations. The questionnaire is created on-the-fly according to the modelled requirements (cf. Fig. 3), which allows updating easily the requirements.



In which social network do you want to deploy the chatbot? (Multi-response)

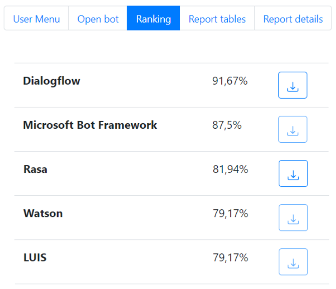
Relevant

☐ Google Assistant	☐ Facebook Messenger	☐ Slack
☐ Viber	☐ Twitter	☐ Twilio (IP)
☐ Twilio (SMS)	☐ Skype	☐ Telegram
☐ Kik	☐ Line	☐ Cisco Spark
☐ Amazon Alexa	☐ Microsoft Cortana	☐ Whatsapp
☐ Cisco Webex	☐ Cisco Jabber	☐ RocketChat
☐ Mattermost	☐ Google Hangouts	☐ Integration with web pages
☐ Own social network		

Do you want to deploy the chatbot on your own host or on the tool host?

Relevant

(a) Requirements questionnaire.



Dialogflow	91,67%
Microsoft Bot Framework	87,5%
Rasa	81,94%
Watson	79,17%
LUIS	79,17%

(b) Resulting tool ranking.

Fig. 5: CONGA recommender support.

TABLE I: Assessment metrics.

	Dialogflow		CONGA	Rasa		
	Back-end	#Files	LOC	Python LOC	Markd. LOC	YAML LOC
Bike Shop[a]	yes	13	80	185	61	187
Mystery Animal[b]	yes	199	7042	9494	13722	879
Smalltalk[c]	no	58	1515	284	1421	281
IoT: Turn lights[d]	yes	6	53	125	23	168

[a] https://bit.ly/38THi8h [b] https://bit.ly/2IQZ8yf [c] https://bit.ly/36KMKrq [d] https://bit.ly/3lEchc2

Each question has a list of options and a selector of relevance.

Fig. 5b displays the ranking of tools ordered by decreasing score. By clicking on the button to the right of a tool, the corresponding code generator is invoked and the developer can download the resulting artefacts.

IV. Evaluation

We have evaluated the migration capabilities of CONGA by importing four third-party, non-trivial Dialogflow agents from GitHub into CONGA, and then generating corresponding chatbot implementations for the Rasa development framework. This evaluation extends the one presented in [6] by considering more challenging bots with back-ends or complex logic, leading to models with thousands LOC in CONGA.

Table I shows size metrics of the chatbots in Dialogflow, CONGA and Rasa. Bike Shop schedules appointments for a shop; Mystery Animal is a guessing game via Q&A; Smalltalk is a chitchatting agent; and IoT turns the lights on/off via NL.

CONGA was able to automatically migrate all chatbot logic from Dialogflow to Rasa, obtaining functional bots. The largest bot parsed into $>$7000 CONGA LOC, and produced a Rasa implementation with $>$9000 Python LOC and $>$14000 LOC in configuration files. This proves the usefulness of our tool.

However, two aspects required manual intervention. First, Smalltalk uses emojis, currently not supported by CONGA. Second, three Dialogflow agents had back-ends developed using Google libraries tightly integrated with Dialogflow. Those cases required configuring the Google services manually and, in one case, implementing a middleware. Generally, the chatbot/back-end connection cannot be migrated fully automatically since it may rely on native technologies of the chatbot platform (e.g., Google's cloud, AWS services).

V. Related Work

The raising popularity of chatbots has led to new tools for their construction (see [3] for a survey). Most are frameworks or platforms, and only a few provide DSLs. The closest work to ours is the model-based solution Xatkit [5]. This provides a textual DSL for chatbot development, but contrary to CONGA, the defined chatbots are executable by providing an execution engine. Moreover, even though Xatkit can help addressing challenge 2 in the introduction (chatbot design), it neither provides a neutral language nor supports tool recommendation or migration (challenges 1 and 3).

Baudart et al. [7] propose an embedded DSL to define Watson chatbots based on an OCaml library, and orchestrate the dialog using ReactiveML. However, an embedded DSL makes the chatbot design less explicit, and while the approach is generative, it is limited to Watson and does not support migration. Protochat [8] provides a graphical DSL for conversation design, and supports a crowd-testing approach whereby crowd workers can provide feedback on the conversation. Finally, some approaches automate chatbot construction from existing artefacts, such as web sites [9].

Overall, there are previous proposals of DSLs for chatbot design, but CONGA is unique for being designed from an analysis of 15 chatbot development tools, and because it addresses tool migration and recommendation.

VI. Conclusions and Future Work

This paper has presented CONGA, a model-driven solution for forward and backward chatbot engineering, featuring a recommender system that assists in selecting the most suitable chatbot development tools. Our approach is extensible by implementing interfaces to create new code generators and parsers, but we are currently working in extension points to facilitate this extensibility. Finally, we plan to conduct a user study to assess the usability of CONGA.

Acknowledgment

Work funded by the Spanish Ministry of Science (RTI2018-095255-B-I00) and the R&D programme of Madrid (P2018/TCS-4314).

References

[1] A. Shevat, *Designing bots: Creating conversational experiences*. O'Reilly, 2017.

[2] P. B. Brandtzæg and A. Følstad, "Why people use chatbots," in *INSCI*, ser. LNCS, vol. 10673. Springer, 2017, pp. 377–392.

[3] S. Pérez-Soler, S. Juárez-Puerta, E. Guerra, and J. de Lara, "Choosing a chatbot development tool," *IEEE Software*, vol. in press, 2020.

[4] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in chatbot development: A study of stack overflow posts," in *MSR*. ACM, 2020, pp. 174–185.

[5] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: A multimodal low-code chatbot development framework," *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.

[6] S. Pérez-Soler, E. Guerra, and J. de Lara, "Model-driven chatbot development," in *ER*, ser. LNCS, vol. 12400. Springer, 2020, pp. 207–222.

[7] G. Baudart, M. Hirzel, L. Mandel, A. Shinnar, and J. Siméon, "Reactive chatbot programming," in *REBLS@SPLASH*. ACM, 2018, pp. 21–30.

[8] Y. Choi, T. K. Monserrat, J. Park, H. Shin, N. Lee, and J. Kim, "Protochat: Supporting the conversation design process with crowd feedback," in *CSCW*. ACM, 2020, pp. 19–23.

[9] P. Chittò, M. Báez, F. Daniel, and B. Benatallah, "Automatic generation of chatbots for conversational web browsing," in *ER*, ser. LNCS, vol. 12400. Springer, 2020, pp. 239–249.