LowcoBot: Towards Chatting With Low-Code Platforms

Francisco Martínez-Lasaca[1,2], Pablo Díez[2], Esther Guerra[1] and Juan de Lara[1]

[1]*Universidad Autónoma de Madrid, Spain*
[2]*UGROUND, Madrid, Spain*

Abstract

Low-code platforms are gaining momentum, allowing the creation of complex applications directly from the browser. This allows their use by individuals with a wide range of backgrounds, but they pose some problems. On the one hand, newcomers may want to grasp the platform capabilities or pinpoint where can they access some functionality, which may prove difficult without guidance. On the other hand, their navigability can be complex: their functionality can be distributed across many webpages and user interfaces, each managing different concepts. Additionally, users may deem more convenient addressing some tasks using natural language instead of navigating visual interfaces. For these reasons, we introduce LowcoBot, a model-driven solution to generate LLM-based chatbots out of low-code platform design models. We demonstrate its capabilities by generating a chatbot for Dandelion+, a low-code platform built within an industrial context, and showing the range of tasks it supports out-of-the-box.

Keywords

Low-code platforms, Chatbots, Code generation, Model-driven engineering, Large language models

1. Introduction

The foreseeable shortage of software developers in the industry is stimulating the seek for alternatives to traditional programming. One of these solutions is *low-code platforms*: cloud environments where the so-called *citizen developers* (i.e., tech-savvy users without an IT background) can create applications thanks to the replacement of code by visual programming [1]. While these platforms strive to provide user-friendly environments, they can be daunting for newcomers unfamiliar with their capabilities or while working with convoluted projects. For these reasons, we posit that integrating *conversational assistants* (i.e., *chatbots*) into low-code platforms can improve their navigability by providing users with more ways to interact with them – see Figure 1 for an example.

In order to develop such chatbots, user requests should be interpreted and responded appropriately in natural language. Current *large language models* (*LLMs*), such as OpenAI's GPT[1] or Google's Gemini[2] are suitable candidates for this task, demonstrating high degrees of compre-

First Workshop on Large Language Models For Model Driven Engineering (LLM4MDE 2024)
francisco.martinezl01@estudiante.uam.es (F. Martínez-Lasaca); pdiez@uground.com (P. Díez); esther.guerra@uam.es (E. Guerra); juan.delara@uam.es (J. d. Lara)
0000-0003-4384-170X (F. Martínez-Lasaca); 0000-0001-8775-4451 (P. Díez); 0000-0002-2818-2278 (E. Guerra); 0000-0001-9425-6362 (J. d. Lara)

[1]https://openai.com/gpt-4
[2]https://gemini.google.com

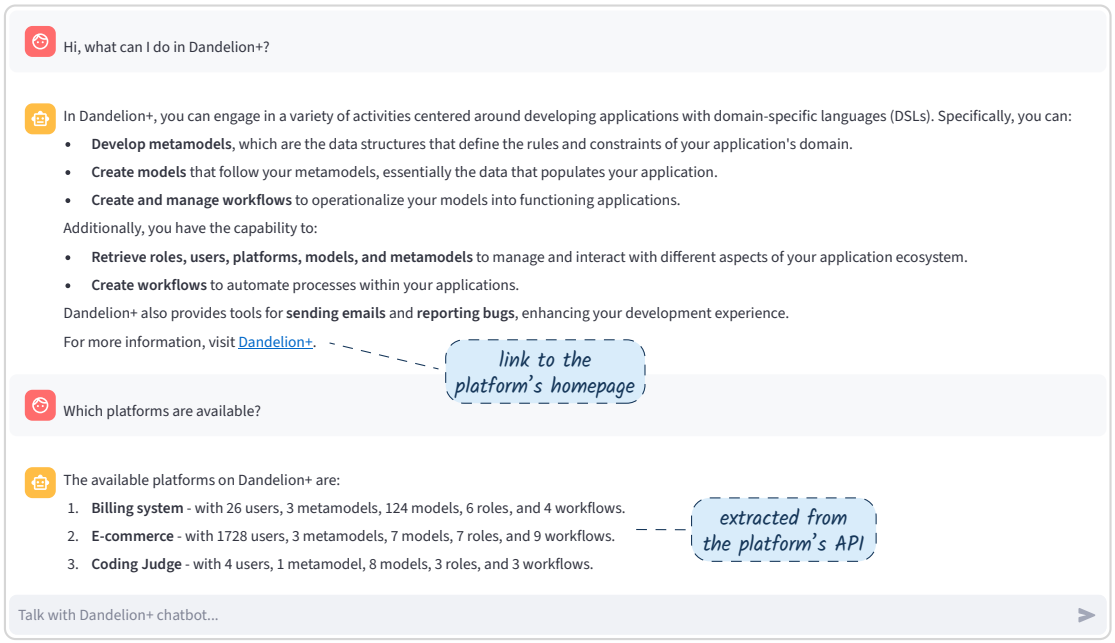

Figure 1: A chatbot generated by LowcoBot for the Dandelion+ low-code platform.

hension over a wide range of user intents [2]. However, interfacing a low-code platform is not straightforward: it requires a *graphical*, *programmatic*, and *ontological* understanding of it. In other words, the chatbot must be knowledgeable about the platform's web interface (the frontend), its API, and its native concepts (e.g., 'user', 'role', or 'project'). In order to represent all this information in a structured and technology-agnostic manner, LowcoBot follows a *model-driven engineering* (*MDE*) approach [3] that allows generating chatbots automatically. As a first step towards its validation, this paper presents an application of LowcoBot by building a chatbot for Dandelion+, an industrial model-driven low-code platform currently under development. The source code of LowcoBot is open source and is available on GitHub.[3]

The rest of the paper is structured as follows: Section 2 presents our approach, Section 3 describes LowcoBot's architecture, Section 4 showcases its application with a chatbot for Dandelion+, Section 5 revises related work, and Section 6 concludes.

2. Approach

Figure 2 presents the steps of our approach, together with the involved components that comprise a chatbot for a low-code platform.

The user interacts with the chatbot in a *chat* interface, consisting of a textual input and the conversation history. To produce sensitive textual responses to user requests, the chatbot makes use of a *large language model* (LLM). Moreover, as native cloud environments, low-code platforms expose *REST APIs* to operate with them via HTTP requests. These two components,

[3] https://github.com/LowcoBot/lowcobot

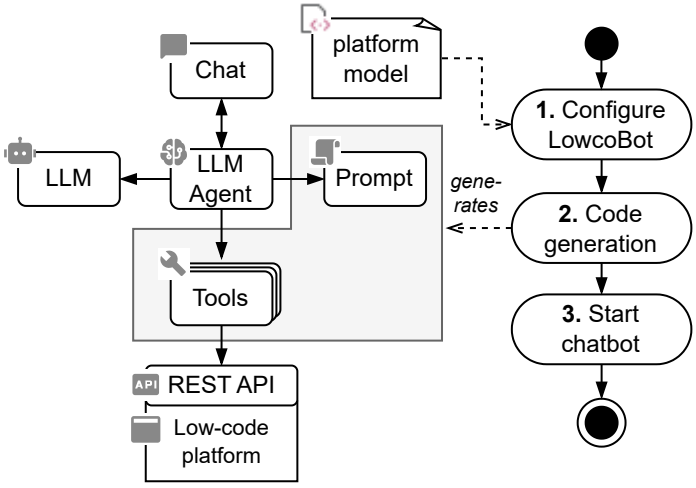

Figure 2: Overview of the approach.

however, must be mediated by an *LLM agent* to accomplish useful responses. As the "heart" of the chatbot, the agent must know the internals of the low-code platform, and allow the LLM to make use of the REST API's endpoints to resolve the petitions.

To make the LLM agent behave as expected (i.e., so it answers platform-related questions), the context of the LLM must be augmented. In this work, we make use of the initial prompt and custom tools to do so. On the one hand, the *initial prompt* is a textual artifact that is fed initially to the LLM to instruct it how to behave. In particular, it should be crafted with the appropriate context so that the LLM can provide satisfying completions for the expected intents of the chatbot. On the other hand, *tools* allow the LLM to execute arbitrary code specified on the chatbot side. Typical LLM tools include code interpreters, arithmetic calculators, or weather forecast providers. In the context of low-code platforms, tools serve as the foundational blocks bridging the LLM and the low-code platform's API.

While the chat interface, the LLM, and the REST API can be easily implemented or consumed, the contents of the LLM agent vary between platforms. However, tools and prompts can be automatically derived from configuration files, making them suitable for automation. LowcoBot exploits this fact to provide a *code generation* solution, taking a model-driven engineering approach, to automatically populate tools and the initial prompt from a low-code platform model specification. Once these are generated, they are wired with scaffold code, resulting in a functional chatbot. As the case study (Section 4) will show, this approach yields a 10× improvement versus manual coding in lines of code.

The next subsections explain LowcoBot meta-model, and the generated prompt and tools.

2.1. LowcoBot meta-model

The meta-model of LowcoBot aims to capture the relevant elements in a low-code platform to produce a conversational chatbot for it. In particular, it focuses on three distinct but interwoven technological spaces – *graphical*, *programmatic*, and *ontological*:

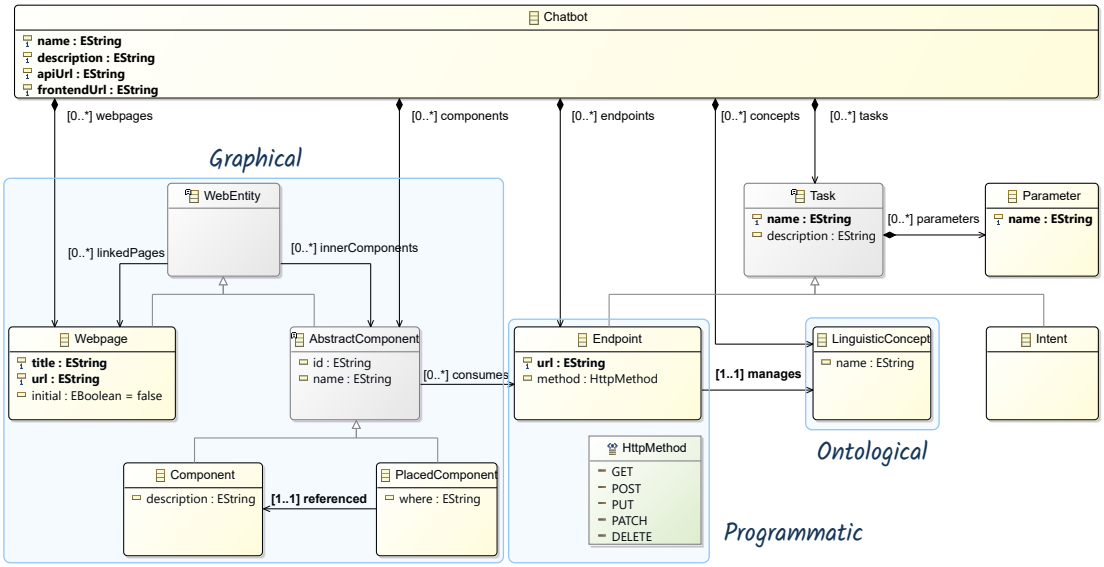

Figure 3: LowcoBot meta-model, relating the graphical, programmatic, and ontological spaces of a low-code platform.

Graphical – the web interface Low-code platforms are web applications that are accessed through a browser and can span across multiple webpages. Being low-code oriented, these platforms rely on visual metaphors (e.g., tables, diagrams, or forms) that are reused in different widgets across webpages. This approach aligns with current web application frameworks, such as React[4] or Angular[5], whose selling point is the encapsulation of web widgets into reusable *components*.

Programmatic – the REST API User operations in the web interface must be reflected on the server side to make them effective. Low-code platforms typically expose their set of functionalities through REST APIs, which can be either consumed by the web interface (thus, being transparent for non-tech users) or by third-party applications. As public interfaces, APIs define the capabilities of a platform: a platform has a capability if and only if it has API *endpoints* covering it.

Ontological – the ubiquitous language Each platform has a different set of *concepts* related in different ways. For example, while both Mendix and OutSystems manage 'roles', they may not be operationally equivalent between both development suites. Moreover, concepts may have different names (signifiers) while pointing to the same conceptual entity (signified) – for example, 'flat' and 'apartment'. Therefore, specialized chatbots for low-code platforms must *speak* the platform's language. This concept is coined *ubiquitous language* in domain-driven design [4], and can be formalized into a *linguistic (meta-)model* that defines the concepts of a domain and their relationships.

[4] https://react.dev
[5] https://angular.io

The resulting meta-model is presented in Figure 3. Note how the meta-model is not exhaustive: it does not model entirely a platform's web interface nor its API or ubiquitous language. Instead, it focuses on capturing the relationships of the entities between the three technological spaces.

Chatbots for low-code platforms have a name and a description, and are determined by the platform's frontend and API URLs. Chatbots encompass webpages, components, endpoints, concepts, and custom tasks. First, *webpages* have a title and are accessible at a certain URL (relative to `Chatbot.frontendUrl`), and can contain components. *Components* describe widgets that appear on webpages. They can either be defined directly as `Components` (with a name and a description) or by referencing other components and "placing" them using a textual explanation with `PlacedComponent.where`. Components can be either nested within other components or embedded in webpages, following the spirit of web components. In turn, components can consume API *endpoints*, which are exposed at a certain URL (relative to `Chatbot.apiUrl`), and have an HTTP method. Endpoints, as providers of the interface of the tool's backend, manage *linguistic concepts* (e.g., 'user', 'role', or 'project'). Moreover, it may be desirable to expose endpoints as tools for solving user *tasks*. Endpoints with `isExposed` set to true are considered for this end. Custom `Intents` can also be specified. Finally, both endpoints and custom intents can specify *parameters* as inputs for their execution.

2.2. Generated artifacts: initial prompt and tools

The previous meta-model can be instantiated in a model that contains the information needed to generate automatically the initial prompt and the tools of a chatbot, for a particular low-code platform.

On the one hand, the *initial prompt* is a textual artifact that instructs the LLM to behave as a chatbot for the low-code platform in question. In particular, the chatbot's name and description specified in the model are injected into it, so the LLM has a preliminary idea of the platform it is interfacing with, even if the platform is initially unknown to the LLM. The initial prompt also contains boilerplate instructions on how to structure its answers, together with a list of the available tools and their parameters. As an example, the following code excerpt presents the first lines of the initial prompt generated for interfacing with the Dandelion+ platform:

```
You are a chatbot for a low-code platform called “Dandelion+”.
A description about you:
    “Dandelion+ is a (meta-)low-code platform aimed at [...]”
Respond to the human as helpfully and accurately as possible.
You have access to the following tools: {tools}. [...]
```

On the other hand, *tools* allow the LLM to execute arbitrary code specified on the chatbot side. Thanks to being exposed in the initial prompt, the LLM can invoke them when deemed appropriate. LowcoBot leverages this to furnish the LLM with the following tools that help navigate, explore, or make use of the low-code platform:

Summary tool It permits answering questions like *What is this?* or *What is ⟨tool's name⟩?* It makes use of the platform's name, description, pages, and tasks to overview its parts.

Figure 4: A LowcoBot model flattened to a graph, as consumed by the *Navigation* tool. Highlighted is the shortest path between the homepage and a `Role` endpoint. The path is computed to answer the question *Where can I change roles?* Legend: blue = `Webpage`, green = (`Abstract`)`Component`, salmon = `Endpoint`, red = `LinguisticConcept`.

Additionally, the LLM is hinted to employ this tool when the user greets the chatbot, so they receive guidance from the beginning.

Navigation tool Exploring new platforms can be daunting for newcomers. For this reason, this tool answers questions like *Where can I ⟨do something⟩ on the platform?* To do so, it generates instructions on how to reach a certain part of the platform's webpage. Internally, the tool constructs a graph as a flattened version of the model, where nodes are webpages, components, endpoints, and linguistic concepts, and edges are any relationship between them. This graph is then consumed by a graph search algorithm to pinpoint the shortest path between the homepage of the platform and the targeted entity (cf. Figure 4).

Capabilities tool Platforms manage different linguistic concepts, each with a different set of capabilities. For example, posts may be created on a social media, but not deleted.

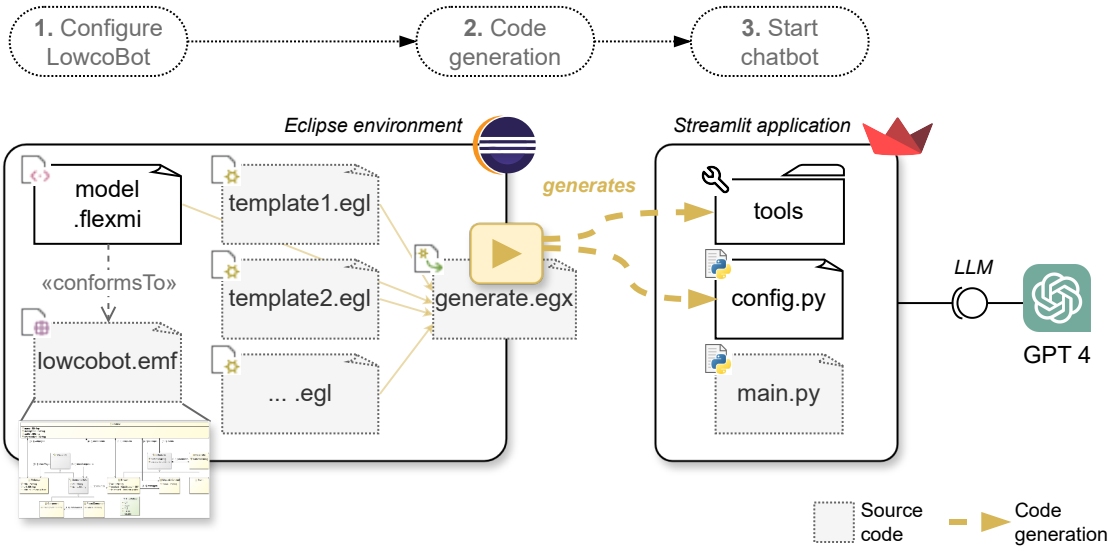

Figure 5: LowcoBot's architecture.

This tool aims at guiding the user when asking *Can I* ⟨*verb*⟩ ⟨*concept*⟩*?*, or *What can I do with* ⟨*concept*⟩*?* If this is possible, the tool will offer the possibility to perform the action and will inform about its location on the platform. Otherwise, it will inform so, while detailing alternative available associated intents or endpoints. To do this, the tool checks the defined endpoints, the linguistic concepts they manage, their HTTP method (e.g., `GET`, `POST`...), and the components (and webpages) where they are consumed.

Intents If the tool has a certain capability, the user expects to be able to use it. In particular, the `Endpoints` (those marked as `isExposed`) and API-agnostic `Intents` promote to tools, usable in the chatbot. Endpoints perform the appropriate HTTP requests when invoked, while agnostic intents delegate their implementation to the chatbot designer. Both can define parameters (cf. *Task* in Figure 3), which result in inputs integrated into the chat to be filled in by the user when the task is triggered (see the last interaction in Figure 6).

3. Architecture

LowcoBot's architecture comprises two technological spaces: the *Eclipse environment* and the *Streamlit application* (see Figure 5).

The chatbot configuration and code generation steps take place in *Eclipse*. In this environment, the user first designs the chatbot in a model conformant to LowcoBot's meta-model. The model is expressed in Flexmi [5] (an XML/YAML-flavoured syntax for model specification), whereas the meta-model uses the Eclipse Modeling Framework (EMF) [6]. Next, this model is consumed in the code generation step, whose source code works atop Eclipse Epsilon [7]. In particular, it comprises several Epsilon Generation Language (EGL) templates and an EGL Coordination Language (EGX) script that orchestrates them. When the transformation is executed, it generates

the chatbot's tools and the initial prompt. Note that, in order to extend LowcoBot with more tools, it suffices to create more EGL templates and link them properly in the EGX script.

The generated artifacts are incorporated into a *Streamlit application*. Streamlit[6] is a general-purpose Python framework for building web applications. In our case, we have implemented a chat interface backed by a structured chat agent[7] defined in *Langchain*[8] – a framework for LLM development. Langchain is agnostic to the employed LLM, allowing switching it easily. Our current implementation uses GPT 4.0, which is available via a pay-per-use API. Langchain also consumes the tools and the initial prompt generated in the previous step directly, as the templates generate Langchain-specific code. Additionally, the *Navigation* tool makes use of the *NetworkX*[9] Python library for graph traversal operations. Finally, Streamlit supports an *embedded mode*, allowing an easy integration into any website via `iframe` HTML elements.

4. Case Study

As a case study, we employ LowcoBot to generate a chatbot for *Dandelion+*. This platform is the evolution of *Dandelion* [8], a model-driven low-code platform for building other low-code platforms aimed at solving industrial problems by the firm UGROUND.[10] Dandelion+ is currently under development.

We model Dandelion+ using 36 entities, including 5 webpages, 10 components, 8 endpoints, 6 concepts, and 2 API-agnostic intents (i.e., sending emails and reporting bugs). This model, expressed in XMI/Flexmi, spans 51 non-empty lines of code (LOC). LowcoBot automatically generates 14 tools[11] and the initial prompt, which add up to 552 Python LOC (without empty lines or comments), yielding a generated artifacts to specification LOCs ratio of 10×. We showcase the features of the generated chatbot in two screenshots.

On the one hand, Figure 1 shows an interaction where the user greets the chatbot and it responds with an overview of the chatbot (*Summary* tool). Then, the user asks for information about a low-code concept, which is answered using an *endpoint intent*. In particular, the tool associated with this last interaction makes a request to the Dandelion+ REST API to retrieve the available platforms in the system and construct an answer.

On the other hand, we demonstrate the rest of the tools in Figure 6, where the chatbot is fully integrated into Dandelion+. First, the user asks for directions on where to perform a specific action within the web interface, and the chatbot responds with indications and links (*Navigation* tool). Then, the user makes a follow-up question regarding the capabilities of a concept, for which the chatbot informs and suggests their execution (*Capabilities* tool). Finally, the user triggers an *API-agnostic intent* (i.e., reporting a bug), which results in a form with the modeled parameters for the user to fill in and submit. Currently, the answer for this intent is fixed. However, given the interest of this scenario from an industrial standpoint, we are working on predicting assigned developers, priorities, and expected resolution times of tickets using LLMs.

[6] https://streamlit.io
[7] https://python.langchain.com/docs/modules/agents/agent_types/structured_chat
[8] https://www.langchain.com
[9] https://networkx.org
[10] https://www.uground.com
[11] Namely: *Summary*, *Navigation*, and *Capabilities*; 8 for endpoints; 2 for agnostic intents; and 1 for intent coordination.

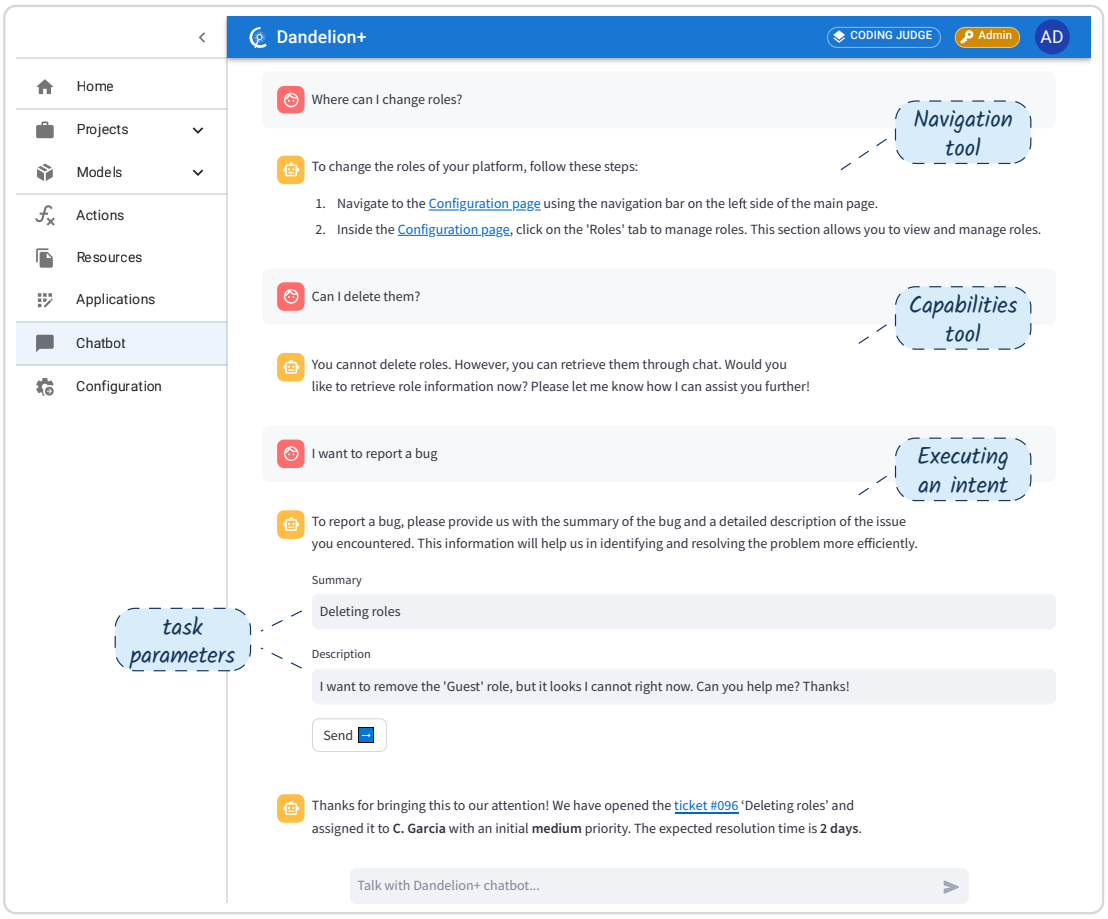

Figure 6: A LowcoBot chatbot integrated into the Dandelion+ platform.

5. Related Work

There have been efforts in the intersection of chatbots, LLMs, modeling, and low-code platforms.

Some works focus on chatbots and modeling. For instance, Socio [9] is a chatbot that produces UML class diagrams from user descriptions of the domain. The chatbot is based on syntactic analysis of the user utterances, using the Stanford parser. Later, the advent of LLMs triggered their use for modeling. For example, Cámara et al. [10] assess ChatGPT for generating UML models from natural language descriptions. Chaaben et al. [11] use LLMs as assistants for model completion, and Chen et al. [12] evaluate different LLMs for automated domain modeling. Tools like Xatkit [13], Conga [14], and DemaBot [15] follow a model-driven approach to designing chatbots. Other works leverage LLMs for low-code tooling. For example, [16] presents an LLM-based tool for designing AI pipelines within a dedicated low-code environment, and in Eclipse GLSP they are currently exploring the potential of AI for improving graphical editors.[12]

[12]See *Enhancing Modeling Tools with AI: A Leap Towards Smarter Diagrams with Eclipse GLSP*

Some approaches combine chatbots with low-code platforms. In [17], chatbots are generated for webpages out of their HTML source code, taking into consideration their semantics, links, and contained information. In contrast, our approach relies on (nested) components and considers the linguistic concepts managed in each component. In [18], Weber uses a node-based editor to orchestrate LLM tools for an IoT-targeted chatbot.

Finally, commercial low-code platforms are integrating chatbots as services for their development suites. For instance, OutSystems[13] and Salesforce[14] can generate chatbots out of customers' application data. Salesforce also supports dedicated intents that trigger internally modeled workflows. Appian[15] covers these use cases, and employs smaller, task-focused chatbots for concrete activities, including model retrieval and diagram construction. Although these chatbots are specifically crafted for each platform, to our knowledge they lack a conceptual understanding of the modeled entities and their relations, and their responses do not include links to help users navigate their platforms.

6. Conclusion and Future Work

This paper has presented LowcoBot, a code generator for chatbots aimed at low-code platforms. LowcoBot is built following a model-driven approach allowing to capture the graphical, programmatic, and ontological sides of low-code platforms. The tool derives automatic prompts and tools that help the user get accustomed to, navigate, and use the platform. We have demonstrated LowcoBot's application to the Dandelion+ platform, generating a set of useful introductory and navigation tools with a generated artifacts to specification ratio of 10× in lines of code.

Currently at UGROUND, we are investigating the potential of model-centric techniques in chatbot construction. We argue that the future of low/no-code platforms lies in chatbots specialized in multiple domains, whose scalability, integration, and ease of development are enabled by a model-driven foundation.

In future work, we plan to support automatic endpoint extraction through Swagger/OpenAPI specifications [19], provide a concrete syntax for the tool configuration, and extend parameters for finer typing. Finally, we want to evaluate the followed approach regarding LLM hallucination, and we intend to introduce defensive mechanisms for when the chatbot is confronted with a question outside its scope.

Acknowledgments

This project is funded by UGROUND, the Spanish MICINN (PID2021-122270OB-I00, TED2021-129381B-C21, RED2022-134647-T), and the EU Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement No 813884.

[13] https://www.outsystems.com/ai
[14] https://www.salesforce.com/eu/products/einstein-ai-solutions
[15] https://appian.com/products/platform/artificial-intelligence.html

References

[1] D. D. Ruscio, D. S. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin?, Softw. Syst. Model. 21 (2) (2022) 437–446.

[2] W. X. Zhao, et al., A Survey of Large Language Models, https://arxiv.org/abs/2303.18223, 2023.

[3] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Second Edition, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2017.

[4] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.

[5] D. Kolovos, A. de la Vega, Flexmi: a generic and modular textual syntax for domain-specific modelling, Software and Systems Modeling 22 (4) (2023) 1197–1215.

[6] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, EMF: Eclipse Modeling Framework, 2nd edition, Pearson Education, 2008.

[7] D. Kolovos, R. F. Paige, F. A. Polack, Eclipse development tools for Epsilon, in: Eclipse Summit Europe, vol. 20062, 200, 2006.

[8] F. Martínez-Lasaca, P. Díez, E. Guerra, J. de Lara, Dandelion: a scalable, cloud-based graphical language workbench for industrial low-code development, Journal of Comp. Langs. 76 (2023) 101217.

[9] S. Pérez-Soler, E. Guerra, J. de Lara, F. Jurado, The rise of the (modelling) bots: towards assisted modelling via social networks, in: Proc. ASE, IEEE, 723–728, 2017.

[10] J. Cámara, J. Troya, L. Burgueño, A. Vallecillo, On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML, Software and Systems Modeling 22 (3) (2023) 781–793.

[11] M. B. Chaaben, L. Burgueño, H. A. Sahraoui, Towards using Few-Shot Prompt Learning for Automating Model Completion, in: Proc. NIER@ICSE, IEEE, 7–12, 2023.

[12] K. Chen, Y. Yang, B. Chen, J. A. H. López, G. Mussbacher, D. Varró, Automated Domain Modeling with Large Language Models: A Comparative Study, in: Proc. MODELS, IEEE, 162–172, 2023.

[13] S. Pérez-Soler, S. Juarez-Puerta, E. Guerra, J. de Lara, Choosing a Chatbot Development Tool, IEEE Softw. 38 (4) (2021) 94–103.

[14] S. Pérez-Soler, E. Guerra, J. de Lara, Model-Driven Chatbot Development, in: Proc. ER, vol. 12400 of *LNCS*, Springer, 207–222, 2020.

[15] B. E. Torres, A. del Río Ortega, M. R. A. de Reyna, DemaBot: a tool to automatically generate decision-support chatbots, https://ceur-ws.org/Vol-2973/paper_278.pdf, 2021.

[16] N. Rao, J. Tsay, K. Kate, V. Hellendoorn, M. Hirzel, AI for Low-Code for AI, in: Proc. IUI, Association for Computing Machinery, New York, NY, USA, 837–852, 2024.

[17] P. Chittò, M. Baez, F. Daniel, B. Benatallah, Automatic Generation of Chatbots for Conversational Web Browsing, in: Proc. ER, Springer, 239–249, 2020.

[18] I. Weber, Low-code from frontend to backend: Connecting conversational user interfaces to backend services via a low-code IoT platform, in: Proc. CUI, ACM, 37:1–37:5, 2021.

[19] H. Ed-douibi, J. L. Cánovas Izquierdo, F. Bordeleau, J. Cabot, WAPIml: Towards a Modeling Infrastructure for Web APIs, in: Proc. MODELS Companion (MODELS-C), 748–752, 2019.