A Blueprint for Integrating Task-Oriented Conversational Agents in Education

Juan Carlos Farah
juancarlos.farah@epfl.ch
School of Engineering
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Basile Spaenlehauer
basile.spaenlehauer@epfl.ch
School of Engineering
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Sandy Ingram
sandy.ingram@hefr.ch
College of Engineering and Architecture of Fribourg
University of Applied Sciences
Fribourg, Switzerland

Denis Gillet
denis.gillet@epfl.ch
School of Engineering
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

ABSTRACT

Over the past few years, there has been an increase in the use of chatbots for educational purposes. Nevertheless, the chatbot technologies and architectures that are often applied to educational contexts are not necessarily designed for such contexts. While general-purpose chatbot technologies can be used in educational contexts, there are some challenges specific to these contexts that need to be taken into consideration. Namely, chatbot technologies intended for education should, by design, integrate directly within online learning applications and focus on achieving learning goals by supporting learners with the task at hand. In this paper, we propose a blueprint for an architecture specifically aimed at integrating task-oriented chatbots to support learners in educational contexts. We then present a proof-of-concept implementation of our blueprint as a part of a code review application designed to teach programming best practices. Our blueprint could serve as a starting point for developers in education looking to build chatbot technologies targeting educational contexts and is a first step toward an open chatbot architecture explicitly tailored for learning applications.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**;
• **Applied computing** → **Interactive learning environments**;
• **Software and its engineering** → *Software system structures.*

KEYWORDS

chatbots, conversational agents, digital education, online learning, task-oriented interactions, software architecture

CUI 2022, July 26–28, 2022, Glasgow, United Kingdom

ACM ISBN 978-1-4503-9739-1/22/07…$15.00
https://doi.org/10.1145/3543829.3544525

ACM Reference Format:
Juan Carlos Farah, Basile Spaenlehauer, Sandy Ingram, and Denis Gillet. 2022. A Blueprint for Integrating Task-Oriented Conversational Agents in Education. In *4th Conference on Conversational User Interfaces (CUI 2022), July 26–28, 2022, Glasgow, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3543829.3544525

1 INTRODUCTION

The presence of chatbots in educational contexts has been steadily increasing over the past few years. Recent surveys [29] have shown widespread interest in the use of chatbots in education both for research and practice, highlighting tangible benefits [25, 31] and promising future applications [8, 27]. More concretely, chatbots have been primarily applied to education as tools for information retrieval or to support language learning [31]. However, a number of challenges limit our ability to integrate chatbots into educational contexts [20], including human-computer interaction (HCI) challenges (e.g., eliciting the uncanny valley effect [5]), technological challenges (e.g., lack of integration with learning platforms [35]), and pedagogical challenges (e.g., difficulty measuring learning outcomes [23]). In this paper, we focus on addressing these last two types of challenges.

First—concerning technological challenges—while frameworks for integrating chatbots in education have been proposed [18, 30, 32], there is no standard approach to building and deploying chatbots for educational purposes. This lack of standard results in either educational chatbots being built from scratch [6, 19] or being powered by technologies not necessarily aimed at education, such as Textit [28], Pandorabots [33], Facebook's Messenger [9], or Google's Dialogflow [17]. Both of these approaches have their limitations. On the one hand, building a custom chatbot architecture requires advanced knowledge encompassing complex computer programming skills [26]. On the other hand, while chatbots built with general-purpose chatbot technologies have been successfully applied to educational settings [31], they often lack the pedagogical scaffolding required to easily design context-dependent dialog flows that can be integrated into a formal learning context. This lack of pedagogical scaffolding highlights the second challenge, which is to ensure that the interaction between the chatbot and the learner is focused on achieving a particular learning goal [23] and stays on

topic [16]—a task that is particularly complex when using free-form dialog and natural language.

To address these challenges, we propose a blueprint for a system that facilitates the integration of conversational agents into online learning applications. Our blueprint is most aligned with work by Griol and Callejas [18], who proposed an architecture for building chatbot interfaces in educative applications and incorporated it into their *Geranium* pedagogical system. Nevertheless, while Griol and Callejas focused on speech-based systems and questionnaire-based learning applications, our aim is to lay out an architecture for text-based inputs, and one that is agnostic to the pedagogical context and type of interaction.

In this paper, we provide an overview of our blueprint. We first outline the design considerations that emerge from the identified challenges and motivate our proposed approach to address these requirements. Second, we detail our blueprint's architecture, defining the key building blocks, components, and processes that make up an interaction between a learner and a chatbot in an educational context. We then present a proof-of-concept implementation of our architecture to showcase how it can support a learning activity. We conclude by discussing how our proposed blueprint could serve as an initial step toward defining an open standard for integrating chatbots into applications aimed at educational contexts. Limitations and future work are also examined.

2 DESIGN CONSIDERATIONS

We build on the aforementioned challenges to define the considerations that will inform the design of our blueprint. The guiding concern was to ensure that these chatbots were (i) integrated and (ii) task-oriented. For each of these aspects, we highlight the requirement that emerges from the literature and our proposed approach to address that requirement in our blueprint.

2.1 Integrated

2.1.1 Requirement. Chatbots should integrate directly with online learning applications [35].

2.1.2 Proposed Approach. Our blueprint is centered on the online learning application that the chatbot is meant to support. This learning application provides the graphical and pedagogical context that will ground the chatbot's interaction. Furthermore, we propose a modular approach that is system-agnostic and does not require any specific technology stack. To maximize reuse, components are loosely coupled and can be composed as needed to support different learning applications. For example, the communication between the learning application and the rest of the architecture is handled by one component in particular. This component could be adapted in order to integrate different learning applications without the need to replace the rest of the architecture. Finally, while we cannot impose that implementations of our proposed blueprint be open source, we strongly recommend it. Open source development facilitates uptake, reuse, continuous improvements, and customization starting from a common codebase, and has been suggested to be favored by developers in education [10].

2.2 Task-Oriented

2.2.1 Requirement. Chatbots should be designed to support learners in achieving well-defined learning goals [23].

2.2.2 Proposed Approach. Our blueprint requires that chatbots be equipped to perform one or more functions. Each function should be pertinent to the context in which the chatbot is deployed. That is, a chatbot's function should take as input the specific learning activity that it is meant to support. To ensure useful and unobtrusive support and minimize the possibility of an unsolicited chatbot interfering with the learning process, a chatbot does not perform its task unless it is summoned by the learner. Furthermore, the chatbot takes into consideration feedback from the learner to improve the way it carries out or communicates the results of its performed function(s). This adaptive approach serves to personalize and enhance interactions with the chatbot and the resulting user experience over time. Privacy considerations related to the way personal data is handled by the chatbot should be transparent to the user.

3 ARCHITECTURE

In this section, we outline the key building blocks, components, and processes that comprise our blueprint, as well as the way they come together to support a learning activity.

3.1 Building Blocks

To illustrate how our architecture fits with the educational context that we are targeting, we define a number of building blocks that make up our components. Specifically, these building blocks show how the chatbot is tasked to support learners during a learning activity by performing one or more specific functions. Here, we outline the key building blocks, providing examples based on the implementation that we will present in Section 4.

3.1.1 Learning Resource. The learning resource constitutes the pedagogical context for the interactions between the chatbot and the learner. The chatbot should be able to perform its task using this learning resource. An example of a learning resource would be a code snippet that can be analyzed for potential issues.

3.1.2 Function. A function is performed by the chatbot given a specific pedagogical context, which serves as the function input. The idea is that the function takes as its main input the learning resource and returns a set of talking points that serve to start the interaction between the learner and the chatbot. An example of a function would be a linter, which is a static analysis tool that can detect bugs and other issues in code [22]. A linter would take as input a code snippet and output the locations in the code snippet where it has detected an issue.

3.1.3 Concept. A concept refers to a subject that the chatbot can identify in the pedagogical context that it is embedded in and that it can subsequently converse about. An example of a concept would be a code styling rule defining how students should name variables. In JavaScript, for instance, the ESLint linter [36] can detect when variables have not been written in `camelCase`. When analyzing a code snippet, the chatbot could detect issues in the code that are related to these concepts and eventually hold an interaction about them with the learner.

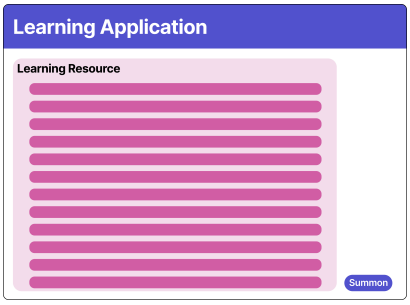

Figure 1: The learning application hosts the learning resource that will be the focus of the interaction and serves as the context for our blueprint.

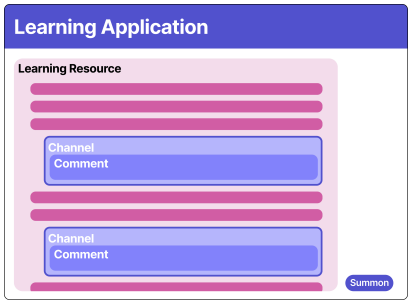

Figure 2: When summoned, the chatbot performs a function and opens channels through which it can hold an interaction with the learner.

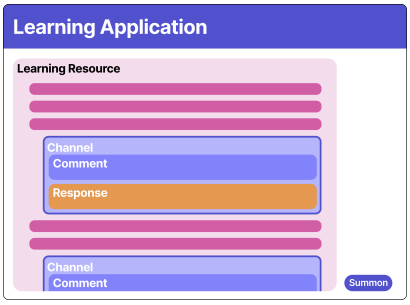

Figure 3: A learner can respond to the chatbot's comments to partake in the interaction that the chatbot has selected for a given channel.

3.1.4 Learning Graph. Each concept has a corresponding learning graph, which represents the states that a learner can be in with respect to the concept at hand. A learner can transition between the different states in the graph by going through an interaction with the chatbot. An example of a learning graph would be a Markov chain with two states (*Don't Know* and *Know*) that indicate whether the learner has understood a concept or not. In its simplest form, this learning graph would not include forgetting, i.e., learners can only transition from *Don't Know* to *Know*.

3.1.5 Channel. A channel is a location in the graphical user interface (GUI) where an interaction between the chatbot and the learner can take place. Channels can be opened by the chatbot after it has performed its function. Once the interaction is complete, the channel is closed. An example of a channel would be a line number in a code snippet. If a chatbot finds an issue in the code snippet, it could open a channel linked to the line number where the issue was located. A dialog box would then appear below the respective line number, allowing the learner and the chatbot to hold an interaction. Figures 1–3 depict how channels can be opened in the context of a learning resource and then serve to hold an interaction between the user and the chatbot.

3.1.6 Feedback. Feedback is a way to signal the quality of an interaction that the learner has had with the chatbot. It can be implicit or explicit. Examples of implicit feedback would be whether the learner engaged with the chatbot at all or whether the chatbot was able to achieve its *intent* (see Section 3.2.1). Examples of explicit feedback would be emoji reactions that the learner can select, a flag to report an issue with a comment made by the chatbot, or a button that the learner can click on to request that the instructor intervene in the interaction.

3.2 Components

Our blueprint's components build on the blocks presented in Section 3.1 to provide the functionalities needed for the chatbot. We divide these components into two types: (i) models and (ii) engines. Models are concerned with keeping the state of the chatbot's interactions with a learner. Engines are concerned with executing the blueprint's processes, which we will outline in Section 3.3. A summary of the components and their building blocks is given in Table 1.

3.2.1 Interaction Model. An interaction takes place within a channel and consists of a natural language dialog that the learner can hold with the chatbot. Each interaction is related to a concept and has an *intent*, which defines the state that the chatbot would like to get the learner to. Our blueprint does not enforce the exact technical implementation of the dialog employed by the chatbot to fulfill its intent. This allows for flexibility in the selection of the scripting and/or natural language processing (NLP) technologies that will power the dialog between the chatbot and the learner. The interaction model also has one or more end states, which indicate that the interaction is over. These end states serve as a way to keep the learner focused on the task at hand. An example of a simple interaction model would be a tree-based script in which the chatbot explains a given concept to the learner and then prompts the learner with questions aimed at verifying whether the learner has transitioned from the *Don't Know* to the *Know* state.

3.2.2 Learner Model. The learner model is meant to summarize a learner's grasp of the concepts that the chatbot is designed to support the learner with. When the chatbot performs its function, it will initialize a learner model. For each concept, it will then select the appropriate active state of that concept's learning graph based on the output of the function. For example, if a JavaScript code snippet contains variables that have not been written in `camelCase`, the resulting learner model generated by the chatbot will contain a learning graph for the `camelCase` concept where *Don't Know* is the currently active state.

3.2.3 Interface Engine. The interface engine is in charge of the connection between the chatbot and the GUI that the chatbot can use to interact with the learners. Some of the functionalities that it handles include: (i) opening the appropriate channels, (ii) posting the chatbot's comments, (iii) exposing the graphical elements required for the learner to respond and provide feedback, and (iv) providing the learner with a way to summon the chatbot.

3.2.4 Task Engine. The task engine is in charge of executing the chatbot's function and translating the output of the function to the

Table 1: An Overview of Our Blueprint's Key Components and Building Blocks Alongside the Functionalities they Support

Component	Building Blocks	Functionalities
Interaction Model	Concepts	Power the dialog between the learner and the chatbot.
Learner Model	Concepts Learning Graphs	Track the state of the *learning graph* for each *concept*.
Interface Engine	Learning Resource Channels Feedback	Provide graphical and pedagogical interfaces to interact with the learning application.
Task Engine	Function Concepts	Execute the chatbot's *function*. Transmit the list of *concepts* to the *knowledge engine*.
Knowledge Engine	Concepts Learning Graphs Interaction Models Learner Model	Create a *learning graph* for each *concept* given by the *task engine*. Create a *learner model* from the *learning graphs*. Manage the *learner model* and the *interaction models*. Update the models when notified. Make knowledge accessible to the *interaction* and *learning engines*.
Interaction Engine	Channel Interaction Model Learner Model	Select the *interaction models* based on the *learning graphs*. Process responses posted to the *channels*. Select the next interaction from the *interaction model*. Notify the *knowledge engine* of possible updates to the models.
Learning Engine	Feedback	Use *feedback* to update the strategies used by the *interaction engine*.

list of concepts that will guide the interactions between the chatbot and the learner.

3.2.5 Knowledge Engine. The knowledge engine keeps track of the state of the learner and the interactions between the learner and the chatbot. That is, the knowledge engine is in charge of updating and tracking the learner and interaction models. Every time a learner response is processed by the interaction engine (see Section 3.2.6), the knowledge engine gets notified of any updates that need to be processed. The knowledge engine also communicates with the interaction engine (see Section 3.2.6) and the learning engine (see Section 3.2.7), providing them with the information they need to carry out their tasks.

3.2.6 Interaction Engine. The chatbot's interaction engine is in charge of selecting the appropriate interaction model for a channel given the state of the learner model. The interaction engine can be configured to follow different strategies and communicates with the learning engine to improve or update these strategies (see Section 3.2.7). The interaction engine is also in charge of processing the learner's interaction with the chatbot. Every time a learner posts a response or provides feedback, the interaction engine processes these actions accordingly and passes the result on to the knowledge engine so that the knowledge engine can update the learner and interaction models.

3.2.7 Learning Engine. The learning engine is in charge of updating the strategies guiding the chatbot's interaction engine based on feedback from the learner. This allows the chatbot to refine its behavior at the end of a session. Over time, the learning engine should allow chatbots to better select the interaction model used to interact with learners based on the learner's learner model.

3.3 Processes

Our blueprint's processes outline the actions that can take place when there is an interaction between a learner and the chatbot. For clarity, we focus on a chatbot interacting with a single learner over a single session. An overview of these processes and how they span the different components of our architecture is shown in Figure 4.

3.3.1 Summon Chatbot. To start interacting with the chatbot, a learner summons the chatbot by asking it to perform its function. This could be done through a button on the GUI of the learning application (see Figure 1). When the learner clicks on this button, the interface engine will notify the task engine that it should start the process of executing the function. The task engine will execute the function and pass the function's result—a list of concepts—to the knowledge engine. The knowledge engine will then create the learner model based on the output of the task engine. This model is then passed on to the interaction engine, which will select the most appropriate interaction model for each of the concepts that the chatbot will tackle with the learner. These interaction models are then passed over to the interface engine, which will open the appropriate channels for the interactions to take place. Figure 2 shows the newly opened channels inside the learning resource.

3.3.2 Hold Interaction. Once a channel is open, the learner can hold an interaction with the chatbot. The interaction is guided by the interaction model selected for that channel and always starts with a message posted by the chatbot. The learner should be able to respond to the initial message through one or more of the following

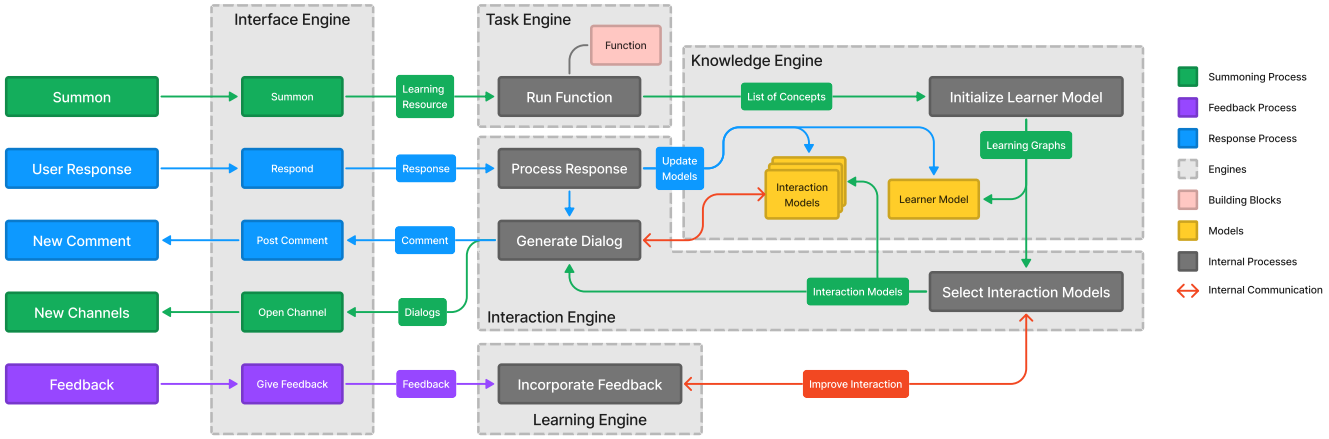

Figure 4: Our blueprint consists of multiple loosely-coupled components, comprising both engines, which are tasked with executing processes, and models, which keep track of the system's state. Here, we illustrate how the different components and processes come together to support the interaction between a learner and a chatbot in the context of a learning activity.

affordances: (i) via natural language through a text input box, (ii) via emoji reactions, and (iii) via quick reply buttons. Text inputs and emoji reactions are commonly available in messaging services and platforms (e.g., WhatsApp [1], Slack [4]), while supporting quick reply buttons has been proposed as a best practice when designing chatbot interfaces [15]. Once the learner responds via any of these affordances (as in Figure 3), the interface engine will forward the response to the interaction engine, which will process the response and select the appropriate next message to post based on the interaction model and its underlying dialog system. The interaction engine will also inform the knowledge engine of the update in the interaction model so that the knowledge engine can update the learner model if the learner has gone through a state transition in the learning graph corresponding to the given interaction's concept. The exchanges between the learner and the chatbot—as well as the corresponding updates—continue until the interaction model reaches an end state.

3.3.3 Provide Feedback. Once an interaction is over, the interaction is analyzed by the learning engine to draw conclusions about the quality of the interaction. This is done by analyzing any explicit or implicit actions that the learner took throughout the interaction (see Section 3.1.6). Based on this analysis, the learning engine provides feedback to the interaction engine. The interaction engine can then use this feedback to change its preferred strategy with respect to a given interaction.

4 IMPLEMENTATION

To provide a proof-of-concept implementation of our blueprint, we integrated it into an online web application used to teach the code review process. We selected the code review process as the context for our proof-of-concept given the increasing presence of chatbots supporting tasks related to code review on social coding platforms [34]. In this section, we present the learning scenario we aimed to support as well as the technical implementation of the chatbot integration. Where relevant, we highlight the building blocks and components presented in Section 3 as they appear in the implementation.

4.1 Learning Scenario

The aim of this implementation was to enhance a code review application used to teach the code review process to software engineering students in tertiary education. Code reviews are an integral part of the software development process. These reviews consist of a developer submitting code for review and then another developer reviewing the submitted code by adding comments to specific lines of code.

The code review application allows an instructor to simulate the code review process within an online learning exercise resembling a computational notebook [13]. Students are shown a code snippet and are asked to analyze it for potential issues. While it is interesting for students to be confronted with an exercise where they can improve their coding practices by detecting errors in code snippets, they may still need some guidance [21]. This need for guidance—which an instructor might not be able to provide to large amounts of students—motivated us to enhance the code review application with chatbots that could support students in completing tasks related to the code review process.

4.2 Technical Implementation

We integrated chatbots into the code review application following the architecture presented in Section 3. The code review application exposes a code snippet (i.e, the *learning resource*) potentially containing some issues (i.e., *concepts*) that can be highlighted by the chatbot. The chatbot's task is to support the learner in understanding those issues by detecting them in the code snippet and explaining how to address them. For this implementation, the chatbot is equipped with a JavaScript linter (i.e., the *function*) able to analyze the code and detect those issues. For example, three issues that can be detected by the chatbot's function are (i) not following

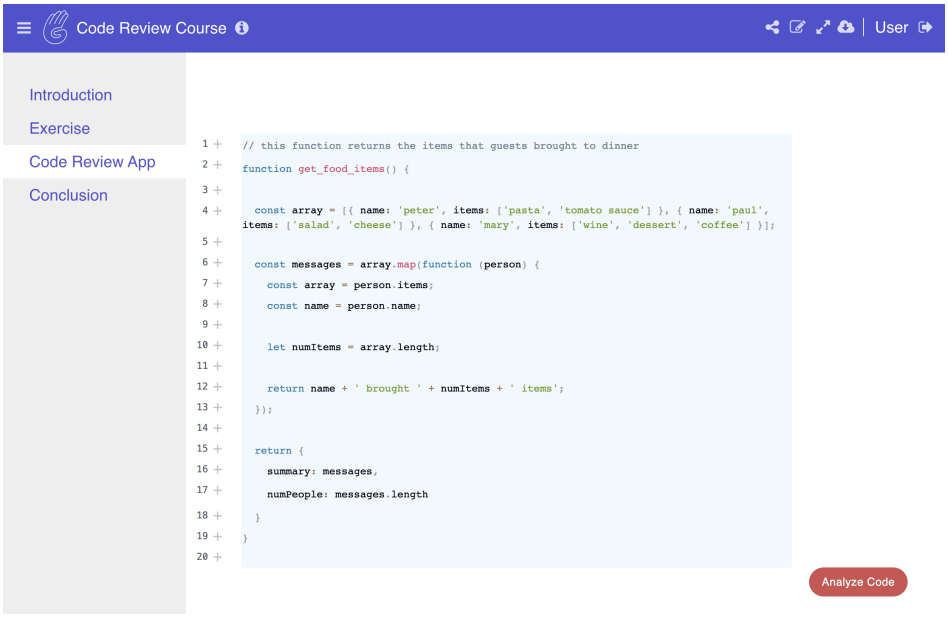

Figure 5: The learning activity focuses on a code snippet that is either written by the student or provided by the instructor via the code review application.

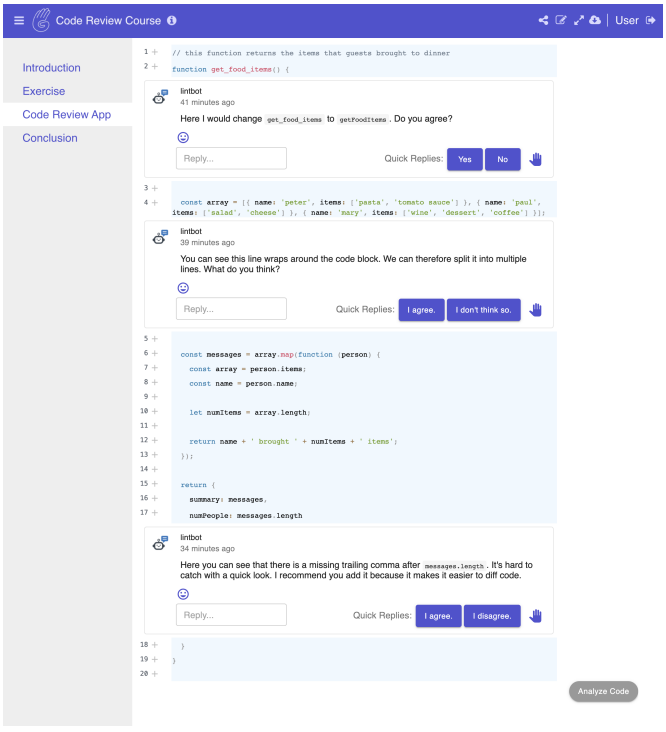

Figure 6: When the learner clicks on the *Analyze Code* button, the summoning process is executed. The chatbot then opens interaction channels below the lines of code exhibiting issues that were detected by the chatbot's linting function.

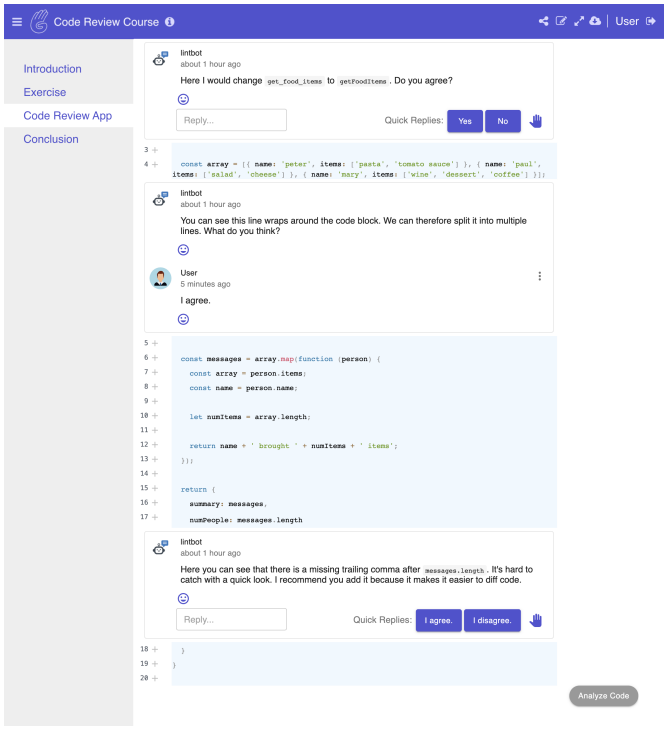

Figure 7: Once the channels are open, the learner and the chatbot can hold interactions about the issues that were detected. These interactions follow a rule-based script.

JavaScript naming conventions, (ii) writing a line of code that contains too many characters, and (iii) not including a trailing comma where possible (see Figure 6). Each issue is associated with a simple *learning graph* in which a learner can either understand the issue or not.

As shown in Figure 5, the learner can start the exercise by submitting a code snippet or using a snippet predefined by the instructor. Whenever a learner is ready to analyze the code snippet, they can summon the chatbot by clicking a button. This button will start the *summoning process*. The *interface engine* will fetch the code snippet and send it to the linter for analysis. The linter will output a list of issues present in the code snippet along with the lines that they appear in. This list is received by the *knowledge engine*, which creates a *learner model* comprising the list of issues detected, each with a learning graph initialized to the *Don't Know* state. The *interaction engine* then randomly selects one of two *interaction models* available for each of these concepts. Both interaction models consist of a short rule-based script—each with slightly different wording—with the *intent* being to transition the learner from the *Don't Know* to the *Know* state. The interaction models are then passed back to the interface engine, which opens a *channel* for each of these interactions to take place under the line containing the respective issue, as illustrated in Figure 6.

The learner can now hold an interaction with the chatbot through any of the channels that have been opened. At each step in the interaction, the learner can respond to the chatbot using either a free-form text response, one of nine preselected emoji reactions, or a quick reply button with predefined text (see Figure 7). The interaction model interprets the response to advance the dialog until it reaches an end state. At that point, if the end state is marked as resulting in a transition, the knowledge engine updates the learner model from the *Don't Know* to the *Know* state. The channel hosting the interaction is then closed and the learning engine marks the chosen interaction model as having been successful or not in completing the transition. The exercise continues until the learner completes all the interactions that were opened by the chatbot or the learner closes the window hosting the application.

5 DISCUSSION AND CONCLUSIONS

The blueprint proposed in this paper is designed to address the two main challenges highlighted in Section 1, namely to have chatbots be directly integrated into learning applications and to provide task-oriented interactions aimed at achieving learning goals. Our blueprint lays out the key building blocks, components, and processes that a system following our architecture would have to include. By abstracting as many details as possible, this blueprint could potentially serve as a first step to implementing a standard architecture for integrating task-oriented conversational agents in education. To test the applicability of our blueprint, we used it to integrate a chatbot into a code review application in order to support students in learning programming best practices. This implementation shows that the building blocks of our blueprint covered the needs of our use case application and facilitated its design and implementation.

Nevertheless, the current version of the blueprint has some limitations that should be highlighted. First, for this paper, we focused on the case of a single learner interacting with a chatbot over the course of one learning activity or session. Further components are required to handle a learner's interaction with the same chatbot over multiple sessions, as well as interactions supporting multiple learners. Second, so far we have only validated the applicability of our blueprint in a code review application. Our aim is to target proof-of-concept implementations in diverse learning contexts and use case scenarios. As chatbots have been particularly useful in computer-assisted language learning (CALL) [2, 7, 31], it would be particularly pertinent to implement chatbots aimed at supporting tasks in CALL (e.g., providing conversational practice, checking grammar, interfacing dictionaries). These implementations will be validated following experimental designs used in previous exploratory studies [11, 12, 14] comprising controlled experiments measuring learning gains, engagement, and usability using both standard (e.g., the User Experience Questionnaire [24] and the System Usability Scale [3]) and bespoke (e.g., scores calculated using the results of pre/post-tests) instruments. Third, the need to support instructors in the configuration and deployment of chatbots built with our blueprint was not addressed. An extension of our blueprint that incorporates mechanisms for configuration, testing, and deployment would ensure that instructors can easily integrate such chatbots into their practice. We aim to further develop our blueprint in order to address these limitations and consolidate its architecture in future work.

REFERENCES

[1] Brian Acton and Jan Koum. 2009. WhatsApp. whatsapp.com

[2] Serge Bibauw, Thomas François, and Piet Desmet. 2019. Discussing with a Computer to Practice a Foreign Language: Research Synthesis and Conceptual Framework of Dialogue-Based Call. *Computer Assisted Language Learning* 32, 8 (2019), 827–877. https://doi.org/10.1080/09588221.2018.1535508

[3] John Brooke. 1996. SUS: A 'Quick and Dirty' Usability Scale. In *Usability Evaluation In Industry*. CRC Press, London, UK.

[4] Stewart Butterfield, Eric Costello, Cal Henderson, and Serguei Mourachov. 2013. Slack. slack.com

[5] Leon Ciechanowski, Aleksandra Przegalinska, Mikolaj Magnuski, and Peter Gloor. 2019. In the Shades of the Uncanny Valley: An Experimental Study of Human–Chatbot Interaction. *Future Generation Computer Systems* 92 (2019), 539–548. https://doi.org/10.1016/j.future.2018.01.055

[6] Fabio Clarizia, Francesco Colace, Marco Lombardi, Francesco Pascale, and Domenico Santaniello. 2018. Chatbot: An Education Support System for Student. In *Cyberspace Safety and Security*, Arcangelo Castiglione, Florin Pop, Massimo Ficco, and Francesco Palmieri (Eds.). Lecture Notes in Computer Science, Vol. 11161. Springer, Cham, Switzerland, 291–302. https://doi.org/10.1007/978-3-030-01689-0_23

[7] David Coniam. 2014. The Linguistic Accuracy of Chatbots: Usability from an ESL Perspective. *Text & Talk* 34, 5 (2014), 545–567. https://doi.org/10.1515/text-2014-0018

[8] Samuel Cunningham-Nelson, Wageeh Boles, Luke Trouton, and Emily Margerison. 2019. A Review of Chatbots in Education: Practical Steps Forward. In *Proceedings of the 30th Annual Conference for the Australasian Association for Engineering Education (AAEE 2019)* (Brisbane, Australia, 2019). AAEE, Barton, Australia, 299–306.

[9] Facebook. 2008. Messenger. facebook.com/messenger

[10] Juan Carlos Farah, Sandy Ingram, and Denis Gillet. 2022. Supporting Developers in Creating Web Apps for Education via an App Development Framework. In *HEAd'22 Conference Proceedings* (Valencia, Spain, 2022). Editorial Universitat Politècnica de València, Valencia, Spain, 893–890.

[11] Juan Carlos Farah, Vandit Sharma, Sandy Ingram, and Denis Gillet. 2021. Conveying the Perception of Humor Arising from Ambiguous Grammatical Constructs in Human-Chatbot Interaction. In *Proceedings of the 9th International Conference on Human-Agent Interaction (HAI '21)* (Virtual Event, Japan, 2021). ACM, New York, NY, USA, 257—262. https://doi.org/10.1145/3472307.3484677

[12] Juan Carlos Farah, Basile Spaenlehauer, Kristoffer Bergram, Adrian Holzer, and Denis Gillet. 2022. Challenges and Opportunities in Integrating Interactive Chatbots into Code Review Exercises: A Pilot Case Study. In *EDULEARN22 Proceedings* (Palma de Mallorca, Spain, 2022). IATED, Valencia, Spain, 10 pages.

[13] Juan Carlos Farah, Basile Spaenlehauer, María Jesús Rodríguez-Triana, Sandy Ingram, and Denis Gillet. 2022. Toward Code Review Notebooks. In *2022 International Conference on Advanced Learning Technologies (ICALT)* (Bucharest, Romania, 2022). IEEE, New York, NY, USA, 209–211. https://doi.org/10.1109/ICALT55010.2022.00068

[14] Juan Carlos Farah, Basile Spaenlehauer, Vandit Sharma, María Jesús Rodríguez-Triana, Sandy Ingram, and Denis Gillet. 2022. Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices. In *2022 IEEE Global Engineering Education Conference (EDUCON)* (Tunis, Tunisia, 2022). IEEE, New York, NY, USA, 1634–1642. https://doi.org/10.1109/EDUCON52537.2022.9766793

[15] María Antonieta Ferman Guerra. 2018. *Towards Best Practices for Chatbots.* Master's thesis. University of Victoria.

[16] Luke K. Fryer, Kaori Nakao, and Andrew Thompson. 2019. Chatbot Learning Partners: Connecting Learning Experiences, Interest and Competence. *Computers in Human Behavior* 93 (2019), 279–289. https://doi.org/10.1016/j.chb.2018.12.023

[17] Google. 2012. DialogFlow. cloud.google.com/dialogflow

[18] David Griol and Zoraida Callejas. 2013. An Architecture to Develop Multimodal Educative Applications with Chatbots. *International Journal of Advanced Robotic Systems* 10, 3, Article 175 (2013), 15 pages. https://doi.org/10.5772/55791

[19] Sebastian Hobert. 2020. Say Hello to 'Coding Tutor'! Design and Evaluation of a Chatbot-based Learning System Supporting Students to Learn to Program. In *40th International Conference on Information Systems (ICIS 2019)* (Munich, Germany, 2019), Vol. 3. Curran Associates, Inc., Red Hook, NY, 1776–1792.

[20] Gwo-Jen Hwang and Ching-Yi Chang. 2021. A Review of Opportunities and Challenges of Chatbots in Education. *Interactive Learning Environments* (2021), 14 pages. https://doi.org/10.1080/10494820.2021.1952615

[21] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Peer Code Review in Higher Education. *ACM Transactions on Computing Education* 20, 3 (2020), 25 pages. https://doi.org/10.1145/3403935

[22] Stephen Curtis Johnson. 1978. *Lint, A C Program Checker.* Technical Report. Bell Laboratories.

[23] Jeya Amantha Kumar. 2021. Educational Chatbots for Project-Based Learning: Investigating Learning Outcomes for a Team-Based Design Course. *International Journal of Educational Technology in Higher Education* 18, 1, Article 65 (2021), 28 pages. https://doi.org/10.1186/s41239-021-00302-w

[24] Bettina Laugwitz, Theo Held, and Martin Schrepp. 2008. Construction and Evaluation of a User Experience Questionnaire. In *HCI and Usability for Education and Work*, Andreas Holzinger (Ed.). Lecture Notes in Computer Science, Vol. 5298. Springer, Berlin, Germany, 63–76. https://doi.org/10.1007/978-3-540-89350-9_6

[25] Alexander Lidén and Karl Nilros. 2020. *Perceived Benefits and Limitations of Chatbots in Higher Education.* Technical Report. Linnaeus University. 49 pages.

[26] Crystal Jing Luo and Donn Emmanuel Gonda. 2019. Code Free Bot: An Easy Way to Jumpstart Your Chatbot!. In *2019 IEEE International Conference on Engineering, Technology and Education (TALE)* (Yogyakarta, Indonesia, 2019). IEEE, New York, NY, USA, 3 pages. https://doi.org/10.1109/TALE48000.2019.9226016

[27] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. 2021. Chatbots Applications in Education: A Systematic Review. *Computers and Education: Artificial Intelligence* 2 (2021), 100033. https://doi.org/10.1016/j.caeai.2021.100033

[28] Nicolas Pottier. 2013. Textit. textit.com

[29] José Quiroga Pérez, Thanasis Daradoumis, and Joan Manuel Marquès Puig. 2020. Rediscovering the Use of Chatbots in Education: A Systematic Literature Review. *Computer Applications in Engineering Education* 28, 6 (2020), 1549–1565. https://doi.org/10.1002/cae.22326

[30] Jonas Sjöström, Nam Aghaee, Maritha Dahlin, and Pär J. Ågerfalk. 2018. Designing Chatbots for Higher Education Practice. In *Proceedings of the 2018 AIS SIGED International Conference on Information Systems Education and Research* (2018). Association for Information Systems, Atlanta, GA, USA, Article 4, 8 pages.

[31] Pavel Smutny and Petra Schreiberova. 2020. Chatbots for Learning: A Review of Educational Chatbots for the Facebook Messenger. *Computers & Education* 151 (2020), 103862. https://doi.org/10.1016/j.compedu.2020.103862

[32] William Villegas-Ch, Adrián Arias-Navarrete, and Xavier Palacios-Pacheco. 2020. Proposal of an Architecture for the Integration of a Chatbot with Artificial Intelligence in a Smart Campus for the Improvement of Learning. *Sustainability* 12, 4, Article 1500 (2020), 20 pages. https://doi.org/10.3390/su12041500

[33] Richard Wallace. 2002. Pandorabots. home.pandorabots.com

[34] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 19 pages. Issue CSCW. https://doi.org/10.1145/3274451

[35] Shanshan Yang and Chris Evans. 2019. Opportunities and Challenges in Using AI Chatbots in Higher Education. In *Proceedings of the 2019 3rd International Conference on Education and E-Learning* (Barcelona, Spain, 2019). ACM, New York, NY, USA, 79–83. https://doi.org/10.1145/3371647.3371659

[36] Nicholas C. Zakas. 2013. ESLint. eslint.org