Data:

Preprint of the paper: *M.J. Rodríguez-Sánchez, Z. Callejas, A. Ruíz-Zafra and K. Benghazi, Combining Generative AI and PPTalk Service Specification for Dynamic and Adaptive Task-Oriented Chatbots*, Accepted for publication at the *22nd International Conference on Service-Oriented Computing (ICSOC 2024), Tunisia.*

Abstract:

In recent years, chatbots have become increasingly prevalent in various business domains, providing services such as booking flights, making hotel reservations, and scheduling appointments. These systems, known as task-oriented chatbots, initiate a conversation to collect the necessary data and subsequently invoke a specific web service to complete the task. Traditionally, they operate on the basis of predefined rules or are trained with specific task data. While effective, this approach is often rigid and lacks adaptability to the evolving peculiarities of individual businesses. For instance, a chatbot designed for general restaurant reservations will request common data such as the number of diners or reservation time, but may fail to accommodate specific preferences such as terrace seating, buffet options, or karaoke availability.

To address the limitations of traditional task-oriented chatbots, we propose an innovative approach leveraging generative AI and a novel service specification concept called PPTalk. This approach enables chatbots to dynamically introduce business-specific elements into conversations, enhancing their adaptability to the unique characteristics of each business. We have developed a proof of concept to demonstrate the feasibility and effectiveness of our proposal, obtaining highly positive results.

1 Introduction

The significance and integration of chatbots in businesses have seen substantial growth in recent years. As users may interact with their own words, they break the communication barriers between companies and their customers, enhancing customer service and operational efficiency [3]. These systems support companies in offering services that are more easily accessible to a broader audience.

Task-oriented chatbots maintain a conversation aimed at fulfilling a particular task, such as booking a flight or making a reservation. Currently, these systems can have dialogue flows configured during the design phase that are either tailored to the specificities of the particular services or designed to provide a general conversation (e.g. for the common aspects related in booking a flight, such as origin or destination). Both approaches have several limitations, including the need for system developers to update the rules or models when services change and the requirement to adapt the system when new functionalities, possibly related to web services, are introduced [1]. Thus resulting in intrinsically closed dialogue systems that cannot invoke services which are not considered during the design phase. This limitation leads to a lack of scalability and makes it impossible to add new functionalities without incurring significant costs in terms of time and money. Moreover, the predefined dialogue flows

in these systems involve asking users the same questions in each interaction, without adapting to the specific requirements of each business's service. For instance, consider a task-oriented conversational system designed for restaurant bookings. Initially, the system is configured to handle reservations for specific dates, times, and party sizes at a given restaurant. However, if the restaurant introduces new dining experiences, such as private dining rooms or special event bookings, the chatbot may not immediately support inquiries related to these new offerings.

Recently, Large Language models (LLMs) have appeared as an alternative to traditional rule and intent-based approaches. As they have been trained with massive amounts of language data, they provide very advanced natural language understanding and generation capabilities, which make them very fitted for question answering and open-domain conversation tasks. However, their usage to develop task-oriented chatbots is still not solved, as it requires that LLMs circumscribe their conversational capabilities to particular domains and are able to invoke specific services.

In this paper we focus on service capabilities to address these issues. Our fundamental contributions are: (1) the definition of a new type of conversational service specification, (2) the definition of a novel process for automatic dialogue generation that invokes a conversational service, (3) a chatbot for restaurant businesses as a proof-of-concept with very positive results.

On the one hand, we contribute the OpenAPI Specification (OAS) named PPTalk, which is used to endow chatbots with the capability to engage in dynamic and adaptive slot-filling dialogues to obtain the necessary pieces of data to invoke the services. On the other hand, we provide a process for the automatic generation of dialogues based on conversational services and LLMs. Unlike state-of-the-art approaches, in our approach LLMs are not the chatbot per se, but are used to improve the main phases involved during conversation: understanding what the user wants to do (intent recognition, e.g., if they want to book a table or cancel a reservation), identifying which parameters need to be requested (e.g., the number of diners and the time), determining when a necessary piece of information has been provided (entity recognition), managing the interaction to decide how to request them (slot-filling), and generating system responses with as much variation as possible through natural language generation.

Thus, our proposal provides further control over the chatbot behaviour. When LLMs are used to manage the interaction in a single step (generating a direct system response for a user input), they constitute a black box; whereas following our process, the conversational interaction between customers and the business is more explainable and manageable.

The rest of the paper is structured as follows. Section 2 presents the related work, Section 3 describes our proposal for the dynamic generation of task-oriented chatbots, Section 4 presents a proof-of-concept and the evaluation conducted. Finally, Section 6 presents the conclusions and outlines future work.

2 Related Work

Previous attempts to automate dialogue management using APIs to enhance the scalability of conversational systems, such as in [9], have resulted in low-quality

dialogues and lack explicit integration into web services. With the emergence of LLMs, chatbots are able to understand more complex inputs, producing more sophisticated dialogues. For example, the ShoppingGPT [12] proposal demonstrates the effective use of GPT models for enhancing product recommendations, improving its ability to understand and respond to user input through multi-turn dialogues. However, challenges include reliance on high-quality training data and risks of generating unsafe or toxic content. Thus, there is a need to investigate approaches that allow to exploit the benefits of LLMs, while retaining control over the generated responses, obtaining better results than traditional pre-trained models like BERT [5].

A first step in this direction is to study prompting strategies to influence the output generated by the model [4]. Creating effective and broadly applicable prompt strategies is complex [14], the reported challenges include the difficulty of choosing and formulating the right instructions to achieve the desired effects. For example, [13] proposes a process to iteratively prototype prompts to enhance user experience, while studies as [10] designs prompts or frameworks [7] [6] to successfully satisfy NLU tasks, for example, slot-filling.

LLMs are best suited for open-domain conversations, adequately adapting them to task-oriented or domain-specific scenarios is still an open research issue. In [8] introduces a domain-specific language (DSL) based on YAML to define task-oriented dialogue system modules, compiled into structured prompts. However, the DSL is specific to the proposal, requiring developer definition and not utilizing existing business service specifications like OpenAPI, resulting in a closed system with non-reusable modules.

The aim of our proposal is to address these open issues by focusing on service capabilities and automated and dynamic dialogue generation based on the services specification. The control over the responses provided by the chatbot, as they are written by the service provider themselves, avoids sending inappropriate responses. Thanks to the use of LLMs as a tool to fulfill NLU tasks such as slot filling or intent recognition, the aim is to achieve good output by appropriately selecting the prompts to be used with the model.

3 Proposal for Dynamic Task-Oriented Chatbot Generation

This section presents our approach for creating dynamic and adaptive task-oriented chatbots. We introduce the Plug, Play, and Talk Service Specification, our extended OAS for chatbots, and the process of automatic dialogue generation that builds the conversation using these specifications.

3.1 Extended OpenAPI Specification for Chatbots: Plug, Play and Talk Service Specification

To seamlessly integrate services with chatbots, we have extended the OAS for the proposed PPTalk services with the elements highlighted in blue in Figure 1.b). These elements are: *1)* the *Question* concept, which contains the questions the chatbot can ask to request a property or parameter, *2)* the *Tag* concept, which allows services to be classified into tags representing business characteristics, *3) DataField*, a conceptual abstraction that encapsulates all data elements

that are either passed to an endpoint method (*Parameters*) or required within a request or response body (*Properties*), and *4) DiscriminativeParameter*, necessary parameters with a `value` property that defines a business characteristic.

A service consists of operations, which can be GET, POST, PUT, or DELETE. For our purposes, we focus on GET and POST operations. GET operations include parameters that may be defined for internal service operations (*InternalParameter*) or are provided by the user (*UserInputParameter*). A *UserInputParameter* can be categorized as *DiscriminativeParameter*. POST operations also contain schemas with properties that must be requested to the user.

3.1.1 Matching the PPtalk Specification and chatbot elements

Figure 1 shows with discontinuous lines how each element in the PPTalk specification metamodel (Figure 1.b) corresponds to elements in the chatbot metamodel (Figure 1.a):

1. **Intent → Operation**: In a conversational system, intents represent the user's goals or desired actions, while operations in a web service specification define the actions the backend service can perform. Formally, let I be the set of user intents and O the set of operations defined in OAS. We define a function IntToOp that maps each intent to a corresponding operation based on their names or synonyms:

$$\text{IntToOp} : I \to O \tag{1}$$

such that

$$\text{IntToOp}(i) = o \quad \text{if} \quad \text{match}(\text{name}(i), \text{name}(o)) = \text{true} \tag{2}$$

where $\text{name}(x)$ returns the name of x, and $\text{match}(a, b)$ is a function that returns true if a and b are the same or synonyms.

2. **Entity → DataField**: In natural language processing systems and chatbots, entities (or slots) are pieces of information required to provide an appropriate response. Similarly, in a web service specification, DataFields represent the values passed to the endpoint method to execute a specific operation, or define specific data points required in a request or response body.

   To perform the matching between entities and datafields, we define a function *EntToD* that is defined formally as follows: let E be the set of entities and D the set of DataFields in the OAS:

$$\text{EntToD} : E \to D \tag{3}$$

such that

$$\text{EntToD}(e) = d \quad \text{if} \quad e \in \text{entities}(d) \quad \text{or} \quad \text{match}(e, d) = \text{true} \tag{4}$$

The function $\text{match}(e, d)$ identifies and extracts relevant information from the input text that corresponds to a DataField d.

$\text{entities}(d)$ is the set defined as follows:

$$\text{entities}(d) = \{e \in E \mid \text{match}(e, d) = \text{true}\} \tag{5}$$

3. **UserTag** $\rightarrow$ **Tag**: In a conversational system, user tags can be conceptually understood as names or adjectives extracted from the user's initial input to identify their preferences, needs, or intentions. These tags help the system understand and categorize user requirements effectively. Tags within the OAS serve as descriptive labels that categorize different aspects or features of services offered by an API.

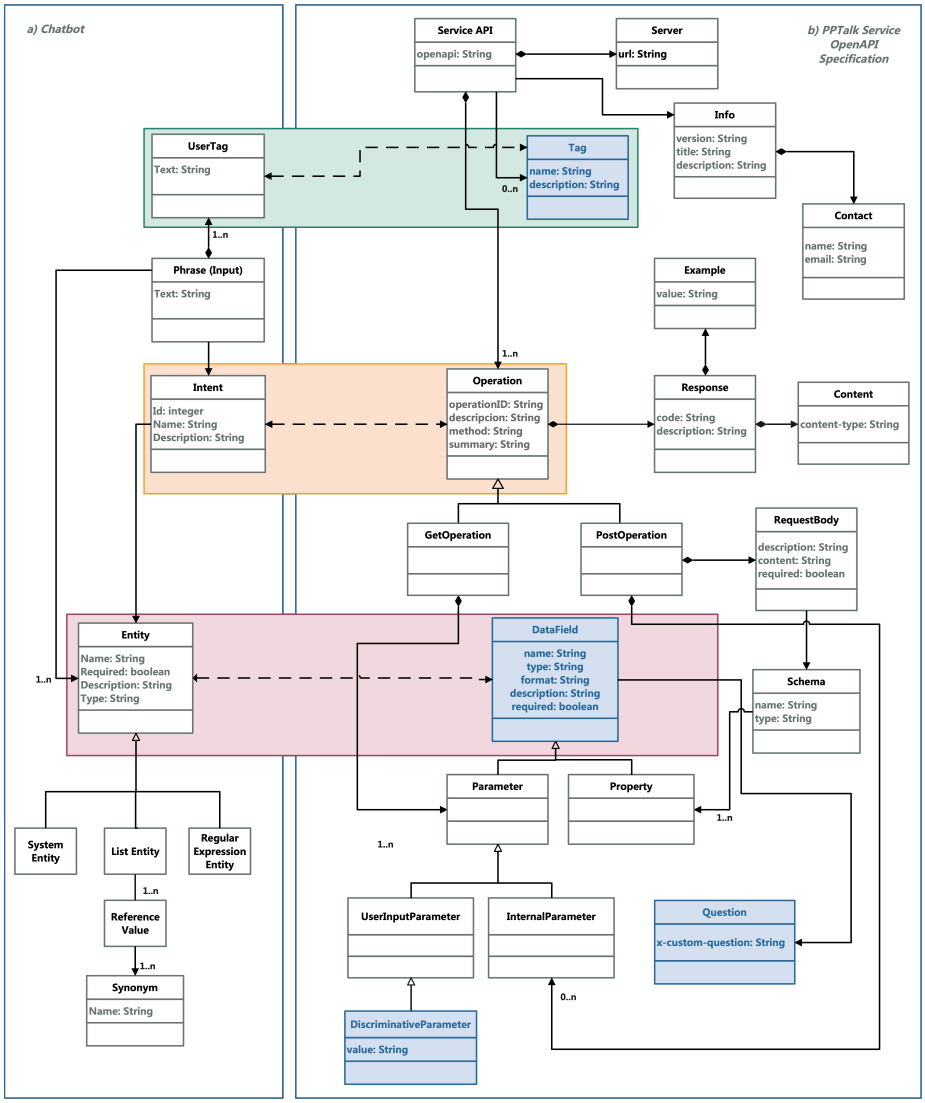

Figure 1: Interconnection between both meta models

Formally, let T be the set of tags within the OAS and UT the set of user

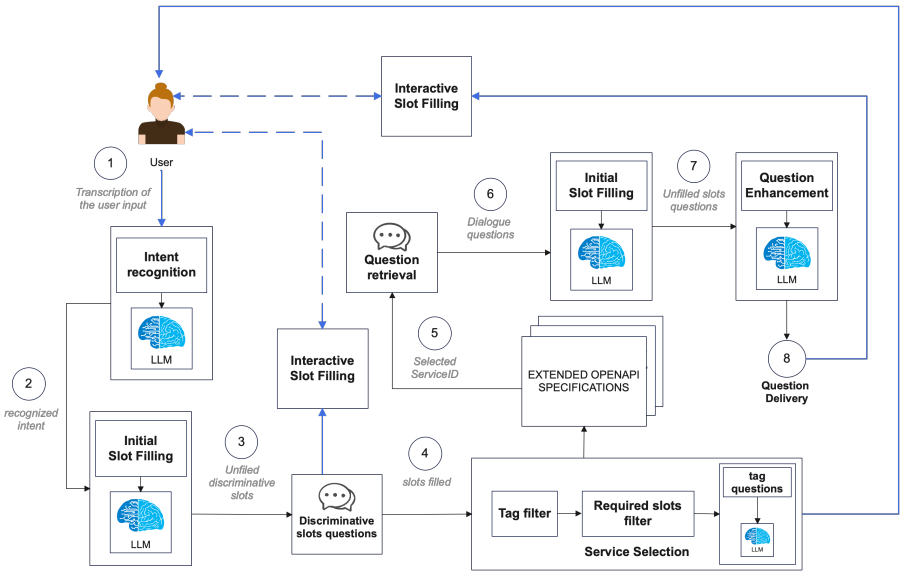

Figure 2: Dialogue building with PPTalk services

tags. We define a function UserTagToTag:

$$\text{UserTagToTag} : UT \rightarrow T \cup \{\emptyset\} \tag{6}$$

such that

$$\forall ut \in UT, \text{UserTagToTag}(ut) = \begin{cases} t & \text{if} \quad \exists t \in T \quad \text{where} \quad \text{match}(ut, t) = \text{true} \\ \emptyset & \text{otherwise} \end{cases} \tag{7}$$

where $\text{match}(ut, t)$ verifies if a user tag matches a tag in the OAS.

3.2 Dynamic Dialogue Building Process

The proposed chatbot building process, shown in Figure 2, dynamically generates and manages the dialogues from PPTalk service specifications following the steps explained below. Some of the phases described (intent detection, slot filling, dialogue enhancement, and question generation) involve prompting a pre-trained LLM. To facilitate understanding, we illustrate each step with an example in which the chatbot responds to the user's input: *"I want to eat in a vegetarian restaurant in Granada."*. Table 1 lists the specific prompts that can be used for this example.

3.2.1 Intent Recognition

When the chatbot receives a request to perform a task, it first detects the user's intent (in our example, the intent is *BookRestaurant*). To do so, it treats intent detection as a classification problem, using the prompt specified in Table 1. Once the *intent* is detected, we perform a string split to identify the verb in the

intent, aiming to find synonyms.

3.2.2 Initial Slot Filling

The slot filling task is performed using the initial user input and the LLM with the prompt specified in Table 1. For the service selection task, it is necessary to fill the discriminative parameters to make the selection as close as possible to the user's preferences. Once the service is selected and the parameter information is retrieved (data fields required to complete a task and corresponding questions), the system performs slot filling to execute that specific operation with the data required by the selected service. Figure 3, highlighted in orange, shows how the initial input fills the discriminative parameter ['foodtype': 'vegetarian'] and the selected service parameter ['area': 'Granada'].

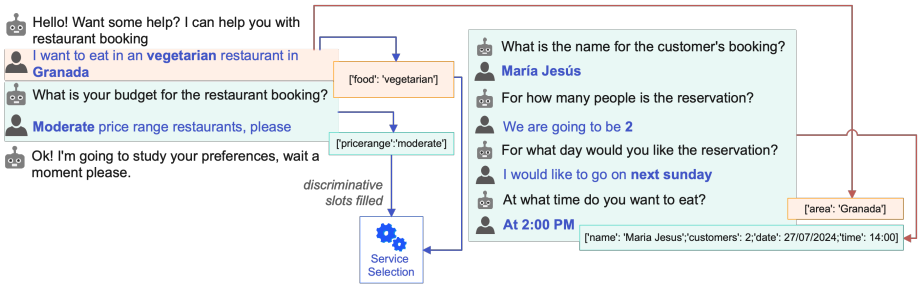

Figure 3: Initial and Interactive Slot Filling

3.2.3 Interactive Slot Filling

All questions are associated with a data field, which in turn is linked to an entity required to perform an operation in the service. When a question or set of questions is sent to the user, a question-response interaction is created. This interaction fills the slots that could not be completed transparently using LLM prompting, as shown in Figure 3 in green colour, allowing the user to perform the slot filling task interactively.

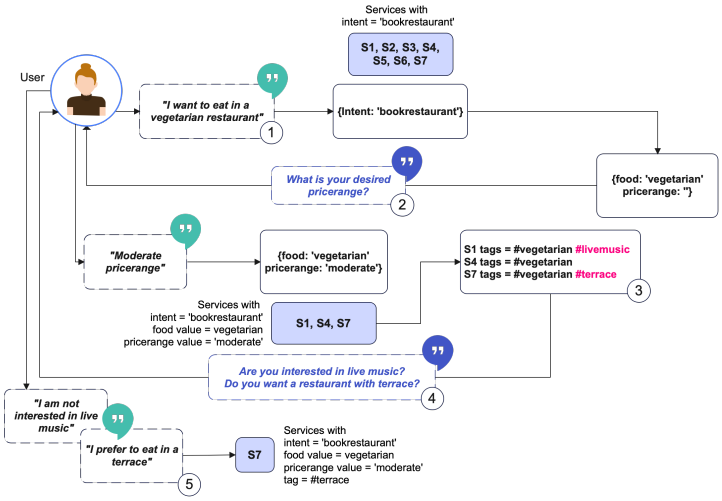

Figure 4: Intent Recognition, Discriminative Slot Filling, and Service Selection Processes

3.2.4 Service Selection

If several services correspond to the discriminative parameters (in the example, several services correspond to restaurants with the same type of food and price range), the system asks the user for their preferences based on other tags (e.g. availability of terrace). This way, the chatbot can select the service that best suits the user's needs following the algorithm specified in Algorithm 1.

These three stages (intent recognition, discriminative slot filling and service selection) are depicted in Figure 4.

Table 1: Chatbot tasks and corresponding LLM prompts.

Chatbot Task	**LLM Prompt**
Intent Recognition	"You are a chatbot in the restaurant domain, and your task is to determine the intent behind a user's input or query. Below is a list of intents related to the restaurant domain: BookRestaurant, RestaurantInformation, FindRestaurant, OrderFood. Given the input '*I*', determine the intent of the user based on the provided intents, return a JSON with only one. Consider that users often want to make reservations when specifying a type of restaurant."
Slot Filling	"Forget the information provided in our previous interactions. Provided the prompt: *P*, these previous inputs during the conversation: *UserAnswers* and the API specification: *PPTalkService*, which contains an endpoint called "*Intent*" with a list of parameters, give me a JSON list with the slots and the values that are given in the prompt directly. If no value given, assign the value NULL. The key of the dictionary is the parameter name and the value is the parameter value."
Discriminative tags question generation	"Provide an informal question that can be answered with yes or no to understand someone's preference regarding a parameter with this tag when selecting a restaurant: "*tagName*""
Question improvement	"Provided the question *Question* in the scope of restaurant booking, give me only one alternative question"

3.2.5 Question Retrieval

The OAS of the selected service returns the questions that must be asked to the user to gather the information required for the specific service (for example, [date, "On what date would you like to make the reservation", time, "At what time would you like to reserve?", diners, "For how many people do you want the reservation?"]).

Algorithm 1 Service Selection Algorithm

Input:
`// Services with matching tags and number of coincidences.`
`tagServices`: Dictionary<ServiceID, Float>
`// Discriminative parameters`
`slots`: Dictionary<String, String>
Output: `selected_services`: List<ServiceID>

$max\_value \leftarrow \max(\texttt{tagServices.values()})$
$max\_keys \leftarrow \{key \mid \texttt{tagServices}[key] = max\_value\}$
$selected\_services \leftarrow []$
`//If there is a tie in the number of matches, the one that best meets user preferences will be chosen.`
if len(max_keys) > 1 **then**
    for service_id **in** max_keys **do**
        $doc \leftarrow \texttt{restaurant\_sv.find\_one}(\{\text{"}\_id\text{"} : \texttt{ObjectId}(service\_id)\})$
        if *doc* **and** 'paths' **in** *doc* **then**
            $params \leftarrow doc[\text{"}paths\text{"}][\text{"}/bookrestaurant\text{"}][\text{"}get\text{"}][\text{"}parameters\text{"}]$
            for param **in** params **do**
                if $param[\text{"}name\text{"}] == \text{"}pricerange\text{"}$ **then**
                $pricerange \leftarrow param[\text{"}schema\text{"}][\text{"}value\text{"}]$
                else if $param[\text{"}name\text{"}] == \text{"}food\text{"}$ **then**
                $food \leftarrow param[\text{"}schema\text{"}][\text{"}value\text{"}]$
                end if
            end for
            if $pricerange == slots[\text{"}pricerange\text{"}]$ **then**
                $selected\_services$.append($service\_id$)
            else if $food == slots[\text{"}food\text{"}]$ **then**
                $selected\_services$.append($service\_id$)
            end if
        end if
    end for
else
    $selected\_services \leftarrow [max\_keys[0]]$
end if
`// If the selected services vector is empty, one service will be chosen randomly.`
if not *selected_services* **then**
    $selected\_services \leftarrow [\texttt{random.choice}(\texttt{list}(\texttt{tagServices.keys}()))]$
end if
return *selected_services*

3.2.6 Question Enhancement

With the help of the LLM, the system detects slots that are already filled and improves the questions for the slots that were not completed with the initial

input to avoid making unnecessary calls to the model and thus make the dialogue less monotonous (for example, as an alternative to the question "On what date would you like to make the reservation?" it suggests "What specific date would you prefer for booking your reservation?").

3.2.7 Question Delivery

The updated set of questions is used to generate the system's response, focusing on slots that have not yet been completed (for example, [date, "What specific date would you prefer for booking your reservation?", time, "What specific time would you prefer to make the reservation?", diners, "How many individuals would you like the reservation to be made for?"]) This sequential process ensures an organized and interactive dialogue between the conversational system and the user. However, the system allows the user to provide values for multiple slots in a single turn (for example, the input "I want a table for two for tomorrow evening" would simultaneously fill the date, time, and diners slots), thus facilitating interaction.

3.3 Dynamic Service Integration

In our approach, web services can be dynamically added to the chatbot during its operation phase. The proposal includes a transformation mechanism through which a standard OpenAPI specification can be converted into a PPTalk specification. Figure 5 illustrates the two ways by which a service provider can add their specifications to the system: 1) uploading an existing OAS specification or 2) creating a new OAS specification. If the service has an OAS specification, it can be uploded and will be prompted to provide the discriminative parameter values and business tags through a simple form.

Using LLM, questions are automatically generated for all parameters with the prompt: *"I am developing a chatbot that users can employ to Intent in the domain Domain. Generate just a question without format that the chatbot can use to ask the user for Parameter"* These questions are included in the resulting PPTalk specification.

If the service does not have an OAS specification, service providers can fill in a form with the service characteristics. A valid OAS is automatically generated through a user-friendly process that requires no prior knowledge, simplifying the creation and integration of PPTalk specifications. The service provider can transform this generated specification into PPTalk using the transformation process.

4 Proof of Concept and Evaluation

To validate the feasibility and effectiveness of our approach, we have developed a proof-of-concept system specifically focusing on a chatbot designed for restaurant bookings within a dynamic shopping mall environment, and conducted an evaluation with simulated users.

Consider a scenario where a shopping mall has a spacious food court that hosts a variety of restaurants, each with specific features that set them apart. For instance, some restaurants offer all-you-can-eat buffets, others have karaoke areas, and some have outdoor terraces.

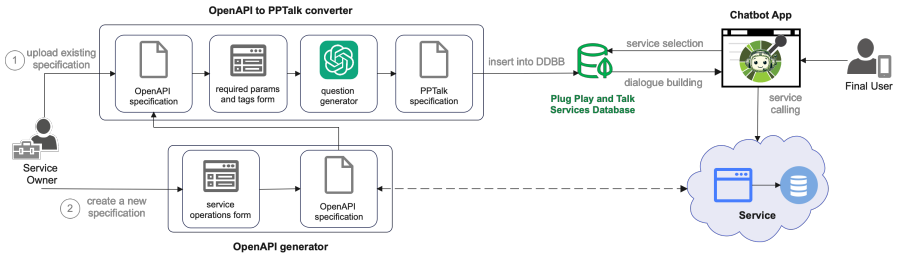

Figure 5: PPtalk Specification addition to the system

Additionally, these restaurants have the ability to evolve and adapt, incorporating new services they did not previously offer. For example, a restaurant might start offering live music to enhance the customer experience, or add a drink service to complement their food offerings. Likewise, over time, new restaurants may open and others may close.

To allow the chatbot to adapt to the specificities of each restaurant and the possible changes introduced in their services, several applications have been developed using Flask and GPT 3.5 as the pre-trained LLM: *1)* a chatbot[1] capable of engaging in a dialogue with users, using one of the PPTalk Specifications integrated into the system; *2)* an application for transforming OAS into PPTalk Specification and managing them[2]; *3)* an API[3] that enables the insertion, editing, deletion, and retrieval of PPTalk specifications from our database, facilitating the efficient management of these specifications.

We have tested this setting with three services[4] that we have developed in Python, and their specifications have been added to the MongoDB of our system, where the PPTalk Specifications are stored. Although the services perform the same action (booking), they are designed to be diverse; for example, two of them allow the client to eat in the terrace, one of them allow pets and has smoking area. Additionally, two of them offer vegetarian food to ensure correct discrimination between them according to user preferences.

Figure 6 shows the example explained in section 3, implemented in an operational chatbot called Chat-PPT. In Figure 6a), the user types the initial input *'I want to eat in a vegetarian restaurant'* and the system, upon detecting the intent (*BookRestaurant*), requests the discriminative parameters that were not collected in the first slot-filling task (in this case the price range). Discriminative parameters (type of food and price range) are used to perform the initial filtering of services corresponding to the selected intent. In our setting, the three services include the *BookRestaurant* operation, and thus correspond to the user's intent. However, only two of them support the selected combination of type of food and price range. Figure 6b) shows how tag questions are asked in order to select one of the two. The questions posed to the user ask about the

[1]https://github.com/mjesusrodriguez/chatbot_mono
[2]https://github.com/mjesusrodriguez/openapitopptalk
[3]https://github.com/mjesusrodriguez/pptalk
[4]https://github.com/mjesusrodriguez/bookrestaurant1, https://github.com/mjesusrodriguez/bookrestaurant2, https://github.com/mjesusrodriguez/bookrestaurant3

properties each service has defined in its tags that may match additional user's needs or preferences. In this case the user wants to eat in a restaurant that has smoking area, so the selected service will be the one shown in the image.

Once the service is selected, the parameters that were not filled in the second slot-filling task, but are required to invoke the service (e.g. the number of diners) are directly asked to the user in what we call *Interactive Slot Filling* as shown in Figure 6c). This process uses the questions written in the service after improving them with the LLM so that they are not posed verbatim as indicated in the specification in every conversation.

Thus, until now we have demonstrated with the development of a functional chatbot the applicability of our proposal, showcasing the adaptability of the chatbot's dialogues to the specificity of the services and also the possibility of integrating new services or modifying existing ones during the chatbot's operation time. To the best of our knowledge, our proposal is innovative and there is no gold standard to compare it with.

Nevertheless, to contribute additional evaluation results, we also underwent an evaluation of the quality of the dialogues generated to show that the chatbot does not ask unnecessary or redundant questions, and that the number of conversation turns is well balanced to identify the users' preferences but at the same time allow an efficient access to services that does not require a lengthy dialogue.

In order to do so, we have developed an automatic method to generate conversations with a simulated user. This is a widely used method in the conversational systems domain in order to optimise the evaluation processes and obtain a comprehensive number of interactions.

This method exploits the pre-trained BERT model presented in [2] that is finetuned for question-answering. This model is adjusted with question-answer data to enhance its performance in extracting answers from a given context or paragraph. In our case, for each conversation, we select as context a random task from the CamRest676 dataset [11], a well-recognized dataset for evaluating task-oriented dialogue systems in restaurant booking domain. Each task in this corpus represents a user objective (e.g. making a reservation at a restaurant with a particular price range). Then, an initial user input is generated to match the context following a variety of scenarios, from more informative inputs providing several entities to more concise phrases that lack important pieces of information.

To generate a response to specific booking questions, such as the phone number, number of diners, or the date and time of the reservation, rules have been established for the simulated user to provide an appropriate answer. Following this method, we have generated fifty conversations between the proposed chatbot and the simulated user.

For each conversation, we record the entire dialogue, the number of turns, the result of the slot-filling task at the end of the dialogue, and the selected PPTalk service for the conversation. Our proposal allows the user to express complex queries that can fill multiple slots in a single turn. For this reason, we have also recorded the number of slots filled with the user's initial prompt, so there will be no need to request this information later.

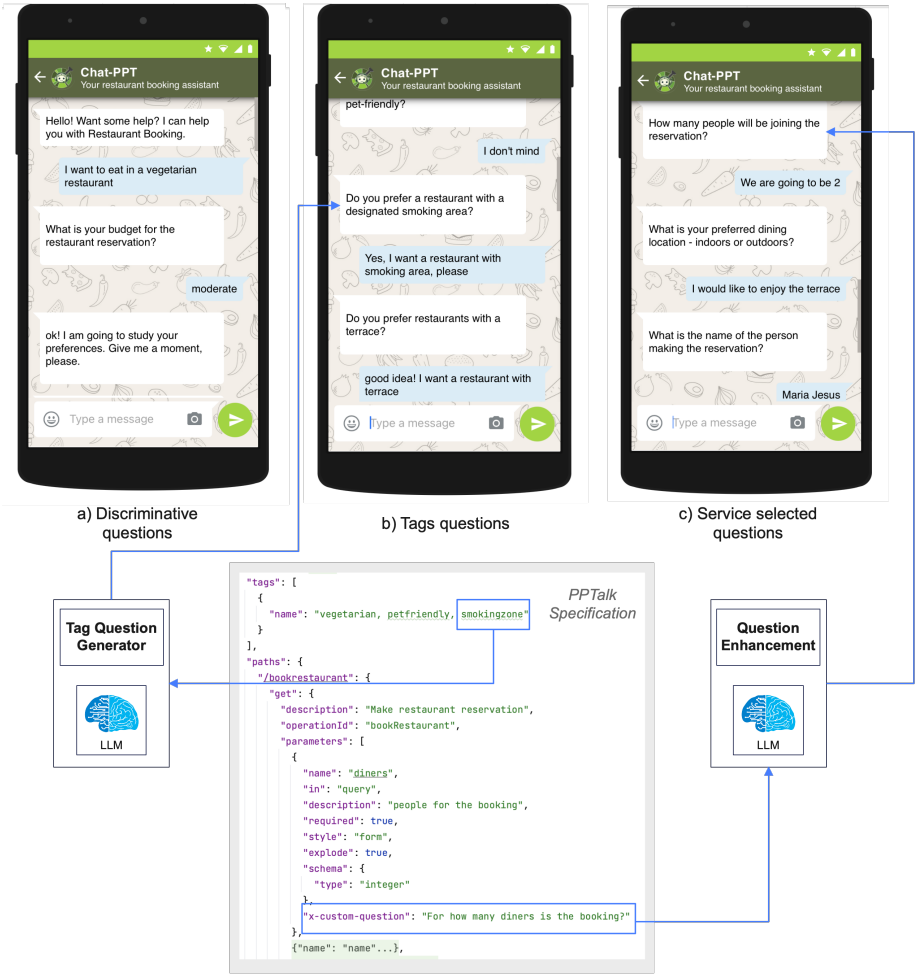

Figure 6: Chat-PPT chatbot with the PPTalk Specification

Table 2 highlights the system's effectiveness in simulated dialogues. The low incidence of unnecessary questions and the absence of redundant queries demonstrate efficient slot-filling with minimal turns (6 to 10), suitable for gathering essential booking details. In the first turn, users fill 30.01% of the slots, enhancing dialogue efficiency. This is partly due to the CamRest676 dataset's context, which does not include specific booking details such as the day or time. The number of slots varies per service, customized for each owner. The F1 score, calculated with the help of a human that fills the slots properly, shows balanced precision and recall, with an F1 score of 78% in correctly identifying instances.

Table 2: Summary of dialogue system evaluation results

Evaluated Aspect	**Result**
Average incidence of unnecessary questions	0.28
Redundancy in questions	None
Average number of turns per conversation	6.06
Percentage of slots filled initially by the user	30.01%
F1 Score	0.78

5 Conclusions

We have presented a novel approach to build dynamic and adaptive task-oriented chatbots combining PPTalk services specification and generative AI.

The PPTalk OpenAPI specification enhances traditional OpenAPI by adding elements like user questions and tags, which facilitate dynamic dialogue generation, as service providers can easily modify services and their elements and chatbots can adapt immediately to such changes. This method allows generating conversations in operation time based on PPTalk specifications, tailoring interactions to user needs and preferences, thereby improving user experience and system efficacy. We also contribute an implementation in the restaurant domain that demonstrates the practicality and effectiveness of our approach, showcasing its scalability and adaptability.

The integration of generative AI with the PPTalk service specification marks a significant advancement in the development of task-oriented chatbots, overcoming many limitations of both traditional rule and intent based systems and one-step extreme-to-extreme LLM based conversations. By exploiting pretrained LLMs in several steps of the dialogue management process, our approach enables the creation of dynamic and adaptive conversational agents that can seamlessly incorporate new services without extensive reprogramming or retraining, while providing more control over the interactions generated.

Future work will focus on extending the application of PPTalk specifications to other domains, fine-tuning LLMs for specific tasks, verifying client-provided data for type and range accuracy, and transitioning to a microservices-based architecture to further improve scalability and flexibility.

Acknowledgments

This publication is part of the R&D&I project GOMINOLA supported by the Spanish Ministry of Science and Innovation (PID2020-118112RB-C21 and PID2020-118112RB-C22), financed by MCIN/AEI/10.13039/501100011033)

References

[1] Brabra, H., Báez, M., Benatallah, B., Gaaloul, W., Bouguelia, S., Zamanirad, S.: Dialogue management in conversational systems: A review of approaches, challenges, and opportunities. IEEE Transac-

tions on Cognitive and Developmental Systems **14**(3), 783–798 (2022). https://doi.org/10.1109/TCDS.2021.3086565

[2] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[3] Følstad, A., Brandtzæg, P.B.: Chatbots and the new world of hci. interactions **24**(4), 38–42 (2017)

[4] Hou, Y., Tamoto, H., Miyashita, H.: " my agent understands me better": Integrating dynamic human-like memory recall and consolidation in llm-based agents. In: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems. pp. 1–7 (2024)

[5] Labruna, T., Brenna, S., Magnini, B.: Dynamic task-oriented dialogue: A comparative study of llama-2 and bert in slot value generation. In: Falk, N., Papi, S., Zhang, M. (eds.) Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop. pp. 358–368. Association for Computational Linguistics, St. Julian's, Malta (Mar 2024), https://aclanthology.org/2024.eacl-srw.29

[6] Labruna, T., Magnini, B.: Addressing domain changes in task-oriented conversational agents through dialogue adaptation. In: Bassignana, E., Lindemann, M., Petit, A. (eds.) Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop. pp. 149–158. Association for Computational Linguistics, Dubrovnik, Croatia (May 2023). https://doi.org/10.18653/v1/2023.eacl-srw.16, https://aclanthology.org/2023.eacl-srw.16

[7] Mok, J., Kachuee, M., Dai, S., Ray, S., Taghavi, T., Yoon, S.: Llm-based frameworks for api argument filling in task-oriented conversational systems. arXiv preprint arXiv:2407.12016 (2024)

[8] Sánchez Cuadrado, J., Pérez-Soler, S., Guerra, E., De Lara, J.: Automating the development of task-oriented llm-based chatbots. In: Proceedings of the 6th ACM Conference on Conversational User Interfaces. CUI '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3640794.3665538, https://doi.org/10.1145/3640794.3665538

[9] Vaziri, M., Mandel, L., Shinnar, A., Siméon, J., Hirzel, M.: Generating chat bots from web api specifications. In: Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. p. 44–57. Onward! 2017, Association for Computing Machinery, New York, NY, USA (2017)

[10] Wei, J., Kim, S., Jung, H., Kim, Y.H.: Leveraging large language models to power chatbots for collecting user self-reported data. Proc. ACM Hum.-Comput. Interact. **8**(CSCW1) (apr 2024). https://doi.org/10.1145/3637364, https://doi.org/10.1145/3637364

[11] Wen, T.H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L.M., Su, P.H., Ultes, S., Young, S.: A network-based end-to-end trainable task-oriented dialogue system. In: Lapata, M., Blunsom, P., Koller, A. (eds.) Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 438–449. Association for Computational Linguistics, Valencia, Spain (Apr 2017), https://aclanthology.org/E17-1042

[12] Yu, R., Guan, Y., Zhan, Y.: Shoppinggpt: A gpt-based product recommendation dialogue system. pp. 501–509 (08 2023). https://doi.org/10.1109/PRML59573.2023.10348314

[13] Zamfirescu-Pereira, J., Wei, H., Xiao, A., Gu, K., Jung, G., Lee, M.G., Hartmann, B., Yang, Q.: Herding ai cats: Lessons from designing a chatbot by prompting gpt-3. In: Proceedings of the 2023 ACM Designing Interactive Systems Conference. pp. 2206–2220 (2023)

[14] Zamfirescu-Pereira, J., Wong, R.Y., Hartmann, B., Yang, Q.: Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. CHI '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3544548.3581388, https://doi.org/10.1145/3544548.3581388