Ontology-based dialogue system for domain-specific knowledge acquisition

Vasile Ionut Iga
Business Informatics Research Center Babes-Bolyai University
Cluj-Napoca, Romania *vasile.iga@stud.ubbcluj.ro*

Gheorghe Cosmin Silaghi
Business Informatics Research Center Babes-Bolyai University
Cluj-Napoca, Romania *gheorghe.silaghi@ubbcluj.ro*

Abstract

Building task-oriented dialogue systems for solving specific tasks within companies represents a challenging research objective especially when the specific expertise of the company is not yet machine-readable formalized. Aiming to collect the specific knowledge in a company-scope knowledge graph, we design the architecture of a dialog system integrating the natural language processing modules with the specific concepts described in the domain ontology and the company's graph of instances. The described system helps growing the knowledge graph with the specific company data, while also providing the NLP building blocks for a future dialog system specifically tuned for the requirements of the target company processes.

Keywords: task-oriented dialogue system, ontology, knowledge acquisition, knowledge graphs

1. Introduction

Nowadays we face an unprecedented growth of chatbots and task-oriented dialogue systems (TOD), all culminating with the tremendously popular ChatGPT[1], as the latest expression of the Artificial Intelligence advances. Chatbots are designed to pursue unstructured conversations on general topics, like human chats [8], being learned on massive open datasets from multiple domains. TOD systems are built to help users solve specific tasks of their interest [6, 9].

While companies from various domains seek to use automated systems for solving repetitive tasks or for helping their employees to better achieve their duties and many of them are reluctant to submit internal expertise to external general chatbots like ChatGPT, there is a high need for TOD systems built on the specific knowledge and expertise of those companies. Before becoming productive within the company, those TOD systems need to ingest the specific knowledge regarding the target tasks.

Semantic Web technologies help organizations to structure their knowledge in a machine-readable format for future usage. Knowledge graphs (KG) became popular as a mean of storing both the conceptual and individual data, delivering advanced reasoning capabilities, in comparison with classical relational databases. Thus, we identify a need for a TOD system to be able to capture the specific expertise in a global KG at the organizational level. Such a KG could be further aligned with open KGs like Dbpedia[2], unlocking the potential of general and specific expertise for building effective TOD systems for organization-specific tasks.

The typical architecture of a dialogue system consists of four modules [8, 16]: Natural Language Understanding (NLU), Dialogue State Tracking (DST), Dialog Policy (POL) and Natural Language Generation (NLG), each of them being responsible for some parts of the processing pipeline. Implementing them could be done in a classical way, with frames and rules or in a

[1]`https://openai.com/blog/chatgpt`
[2]`https://www.dbpedia.org/`

modern fashion, where components of the processing pipeline leverage deep learning models. Recent approaches [1, 13, 15] abandon the pipeline processing approach and propose end-to-end systems, where the response is generated by integrated DL models, properly trained. DL models training could happen from scratch, or could leverage pretrained language models [17] like BERT [7] or GPT3 [3]. Regardless of the selected architecture, if DL models are employed, they should be trained or at least fine-tuned on annotated data. If the TOD system is supposed to help users in fulfilling domain-specific tasks, a way to incorporate domain-specific knowledge into the models should exist. This could be done either by preparing big annotated datasets, or by leveraging already existing knowledge repositories.

Having in mind the final goal of building a TOD system for solving specific tasks within a business organization, in our paper we describe the architecture of a system that, in the end, will accomplish the needs of the company. Our system fulfills two main objectives: (i) to help the company create its global KG with the specific knowledge within the company and (ii) to provide annotated data for future training of deep learning models needed for the production of an effective TOD system. Furthermore, we design the skeleton of the future TOD system and, from the scientific point of view, we leverage the reasoning capabilities of knowledge graphs for enhancing the effectiveness and naturalness of the TOD system.

Our system is built in a classical manner, following the pipeline architecture with four modules. Rather than employing deep learning models as we are missing company-specific annotated datasets, we use the rule-based approach with templates for providing user support with the essential Create-Retrieve-Update-Delete (CRUD) procedures for knowledge management. We integrate an existing knowledge base with the selected classical pipeline architecture of the system. At a higher level, when the knowledge base will include sufficient domain-specific information, we will leverage the constructed datasets for fine-tuning particular NLP models to enhance the performance of the TOD system.

The paper evolves as following. In section 2 we describe related work for building TOD systems with the help of ontologies. Section 3 describes the architecture of our system. Section 4 discusses the pro and cons of our architecture and presents some incipient results. Section 5 concludes the paper.

2. Related Work

The literature models a conversation as successive turns, each turn being a pair of speech acts emitted by the human user and the dialogue system. In our approach we restrict the speech acts only to textual ones. Responding to the user speech acts, the system should be able to detect the start and the end of a conversation. As we intend to make the conversation look as natural as possible, we permit each conversation to have multiple discussion threads, on multiple topics. The general objective of a dialogue system is to generate a response for each user entry. Its specific objective is to help users in accomplishing some requested business procedures.

Similar to us, Nazir et al. [10] build a chatbot for constructing a knowledge base regarding the fashion industry. Using tools like Protege for ontology engineering, SPARQL for querying and Jena for reasoning, they build much more of a query-answering system only able to respond user information retrieval queries. Tuan et al. [14] integrate KB reasoning in a dialogue system, such that, based on the dialogue history, to extend the response given by a transformer model with the help of semantic reasoning. Rather than using the KB inside the dialogue system, they produce the specialized response with the help of the KB. This represents one of the reference results for our approach. Yang et al. [18] use a knowledge graph within the architecture of an end-to-end TOD system. They encode in a KG the information transmitted by the user during the conversation and use this knowledge to generate the response with the help of a GRU DL model. A somehow similar idea is presented by Caudhuri et al.[5] as their objective is to learn the semantic relations required to respond to the user utterances and for that they build a local conversation graph. They work with BERT-encoded tokens and the KG is used to generate the system response. Andreas et al. [1] assign a local graph of actions to each utterance produced by the user and solve this local graph similar to a classical expression evaluation in order to decide about the action the system needs to take. To learn the DL model, they use a dataset with conversations and associated graphs for each utterances in the conversation. Their graphs look rather like parsing trees than knowledge graphs, and the specific knowledge is implicitly learned with the help of the DL from the input dataset. Bordes et al. [2] develop a rule-based system to construct datasets for learning end-to-end models for TOD systems. Their knowledge base is given and could be interrogated using a specific API. As opposed to us, they do not intend to grow the KB, but rather only to generate relevant dialogues for training various DL models.

Research investigated above mainly use the KGs to enrich the natural language response of the system with previously existing information. Departing from them, we employ the KG to map the context of the dialogue, somehow similar with [1], which map the conversation into a graph that is converted into a program, therefore, not a standard KG. Our pipeline architecture presented in section 3 is complemented by both the local conversation-scope KG and the global organization-scope KG, where we persist validated information acquired from the users. Therefore, we aim to prepare the architecture for our future TOD system, to grow the organizational KB and to acquire the requested datasets for DL models training.

3. System architecture

In this section we describe the architecture of our system. The system is written in Python 3.9. The ontology is described in RDF with the help of the Turtle notation. The local KG maintained during each conversation and the global KG are managed with the RDFLib[3] library. The global KG is persisted in an external file at the end of each running experience. The queries towards the local and the global KG are written in SPARQL. We plan to migrate the global KG to a KG server like GraphDB.

3.1. Overview

Without losing the generality, the purpose of the system is to collect information regarding specific instances described with the help of a general domain ontology. The ontology contains concepts and relationships. Each concept is defined by basic properties which could not be expanded anymore or by complex properties which are other concepts. Figure 1 presents a simple domain ontology used through this section to help the understanding of our system.

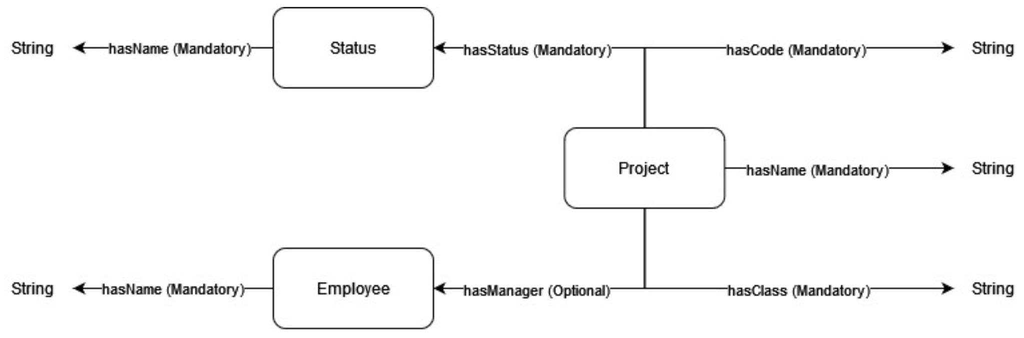

Fig. 1. Domain ontology example

Each relationship in the ontology could be compulsory or not. The system needs to collect all compulsory properties of an instance to consider it well-described. In the domain ontology

[3]https://rdflib.readthedocs.io

example of fig. 1, the Project concept has 5 relationships, two of them being complex ones (hasStatus and hasManager) and one of them (hasManager) not being compulsory.

From a fresh startup, running the system multiple times results in creating a persistent knowledge graph (KG) containing instances belonging to the input ontology. For the actual iteration described in this paper, the system implements the following *procedures*: create (or insert), retrieve (or select), update and delete, well-known under the acronym CRUD. The system interacts with two such knowledge graphs: a general one, which initially provides the domain ontology and a local graph which is grown from zero during each conversation (i.e. system run). The system moves items from the local graph to the general one each time when it identifies the success of a given procedure. Fig. 2 describes the state of the system at the start of a conversation. The local graph maps the context of the conversation, holding all important aspects of it. It has an additional role of an intermediary layer between the user and the general KB, as the TOD system validates the data before persisting it in the general KB.

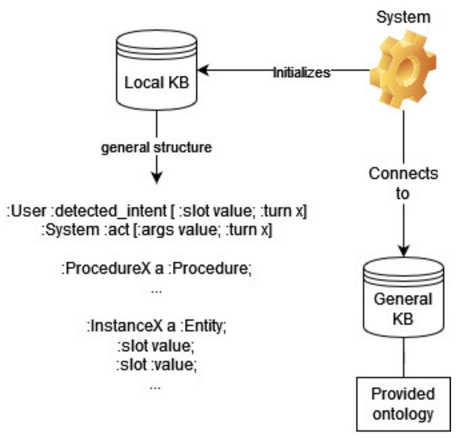

Fig. 2. The state of the system at the start of a conversation

The user starts a conversation with the system. Each conversation is divided in a number of turns, each turn consisting of a pair of user utterance vs. system response. Based on the general KB, the system directs the conversation towards achieving one of the procedures enumerated above. The conversation ends either successfully or by any interruption. During the conversation, each validated procedure is persisted from the local graph to the general KB. Fig. 3 depicts a user-system conversation.

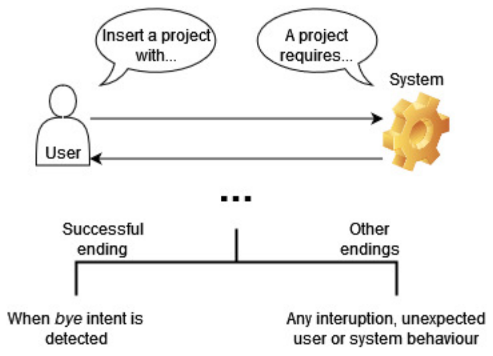

Fig. 3. The conversation flow

Fig. 4 depicts the pipeline architecture of the system, and the main processings done within each module.

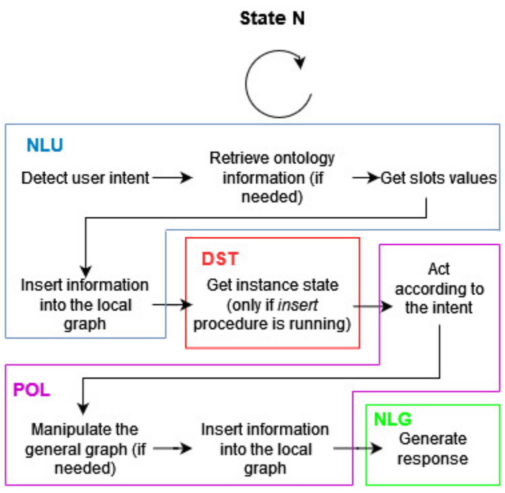
State N
NLU
Detect user intent
Retrieve ontology information (if needed)
Get slots values
Insert information into the local graph
DST
Get instance state (only if *insert* procedure is running)
Act according to the intent
POL
Manipulate the general graph (if needed)
Insert information into the local graph
NLG
Generate response

Fig. 4. The pipeline architecture of the system

3.2. The Natural Language Understanding (NLU) module

NLU module performs the steps described below:

1. detection of the user intent. The following 12 user intents are implemented: *hello, goodbye, thank, insert, select, delete, update, agree, disagree, remove, cancel, switchEntity* and *default*. Some are mapped with the CRUD procedures mentioned above (*insert, select, delete, update*), others help the system starting / ending a conversation (*hello, goodbye*), confirming or not a given response (*agree, disagree*), removing unwanted parameters supplied in the active procedure (*remove*) or signaling the success or abandon of a given procedure (*thank, cancel*). The default intent is used if neither of the above-mentioned intents is discovered in the user utterance. Special attention is given to the *switchEntity* intent which allows to system to navigate through multiple discussion threads. We will discuss this feature in a distinct paragraph, as it represents one essential feature of our system. The 12 user intents were chosen based on two goals: (i) to maintain the natural flow of a conversation and (ii) to enable the CRUD operations on the KB. Therefore, *hello*, *goodbye*, and *thank* fulfill the first objective, by giving the system the capacity to naturally engage in a dialogue. Next, *insert, select, update*, and *delete* map the four available procedures on the KB, the standard CRUD operations. *Agree, disagree, remove, cancel*, and *switchEntity* provide continuity of execution for the discussed procedure. They enable basic actions for the user, in order to drive the conversation's flow in the desired direction. A user can agree or disagree with the system's behavior, may remove information from the dialogue state, such as filters or values to be inserted, can cancel the ongoing process if something is considered wrong, or might switch the thread of discussion, if other threads are available.

2. retrieve information from ontology. The system queries the general KB with respect to the identified intent and returns either existing information about the items in the general KB or schema information from the domain ontology. Specifically, if the detected intent is *insert*, the query over the KB returns the parameters of the specified entity, mapped as properties, their range and their minimum cardinality (which determines if a parameter is mandatory or optional).

3. slot filling: based on the parameters returned in the previous step, the system detects the slot values from the user utterance.

4. insert acquired information in the local KB. The system updates the local graph with the acquired knowledge, under the following form:

```
User :detected_intent [anonymous node].
```

The [] node inserts the current turn together with all its slots and their specific values.

If the user selected to start a novel procedure, the system generates the unique ID of this procedure and inserts the corresponding affected instance in the local graph. Thus, if noticing that the requested intent affects a novel instance and not the currently processed one, the system saves the active instance in a LIFO list and moves the discussion to a novel thread for managing this novel procedure request.

If the user agrees with the action proposed by the system (which could be to insert, remove, or update some instances in the general KG), together with updating the general KG within the POL module, the system will propose the user to continue one of the already started procedures, which are saved in the LIFO list. If the user agrees, the NLU will detect the *switchEntity* intent which will be processed as described in subsection 3.6.

In this way, with the help of the information stored in the local KG and the LIFO list we succeeded to manage multiple discussion threads, as opposed to the standard approach in the literature that usually assumes that each conversation is just a single discussion thread.

3.3. The Dialogue State Tracking (DST) module

DST module only manages the state of the instance being discussed, as opposed to the standard behavior of a DST in the literature [8, 16], where it usually stores the state of the whole system. DST becomes relevant only if an insert procedure is on-going, as it retrieves from the local KG the structure of the instance with the active ID. In our architecture the local graph helps managing the state of the dialog, by storing and managing many threads of discussion in a single conversation, as opposed to the standard approach of pipeline-based TOD systems, where DST is managing only one discussion thread.

3.4. The Policy management (POL) module

POL module decides how the system reacts to the user intent. Here we implemented the majority of the data processing including:

- query the local and general KGs in order to retrieve important necessary information from the dialogue context
- manipulate both graphs if needed. The system might insert, update, delete information from the local or general KGs
- decides about the response action of the system and inserts this information into the local graph, under the form of

```
:System :action [anonymous node].
```

In the [] node, the current turn is inserted, together with all collected information about the parameters.

System actions could be one of the following: *hello, goodbye, welcome, wrongFormatSelect, wrongSelect, showSelect, confirmDelete, wrongDelete, dependencyDelete, confirmUpdate, wrongUpdate, dependencyUpdate, showUpdate, confirm, wrongLiteralDataFormat, cancelProcedure, askStep, reject, chooseReject, removeParams, requireParams, confirmParams, preExistingEntityCancel, chooseEntity, wrongEntity, switchEntity, default*. They were designed to properly respond to all 12 possible user detected intents.

3.5. The Natural Language Generation (NLG) module

NLG module transforms the selected action into natural language. It has pre-built templates for all the above-mentioned actions. In each template, the system can dynamically insert generated information by replacing placeholders from utterances' templates.

Within the NLG module, the utterances of the turn are saved in a local file, together with the corresponding labeling requested in the specific datasets for various NLP tasks of interest in training TOD modules like NER, Intent detection and slot filling, DST, POL and NLG.

The processing steps mentioned in fig. 4 and described below take place at every turn, for producing the system response. Every time that a procedure is validated as being finished, within the POL module the system persists the novel acquired information in the global KB. Therefore the global KB grows, incorporating acquired knowledge from the specific target organization.

3.6. Dealing with multiple discussion threads

The domain ontology is a recursive structure, each concept might be in a relationship with other complex concepts. Thus, to be able to insert or update a given instance, the user needs to transmit specific information also about the other instances being in relationships. Therefore, the system should permit the user that, in the middle of a conversation for solving a procedure related to a given instance, to stop and move along the conceptual graph and starts a novel procedure regarding another instance in relationship with the first one. To solve this request, a new discussion thread is instantiated and a novel node is inserted in the local KG as part of step 4 of NLU. The ID of the active instance is inserted in a LIFO-style list and this new node becomes the one active. In future turns, when the system ends the procedure started regarding the new active node, it discards it in the general KG and it proposes the user to resume one of the procedures found on the LIFO list. The user could select any of the procedures in that list and the selected procedure, together with its instance ID becomes active. If such a selection is done by the user, the system will detect within NLU the special intent called *switchEntity* and continue processing one of the procedures started before the one that is currently finishing.

4. Discussion

In this section, with a running example, we discuss and argue the rationale for our design decisions. To better understand the functioning of the modules presented in section 3, an example conversation is depicted in table 1.

At the very beginning the user says "insert a project...", the *project* concept will be discovered on the general KG and the system will perform the steps provided in the NLU module. Specifically it will perform the following steps: (i) determine the insert intent, (ii) obtain the specific parameters of the discovered concept from the ontology, (iii) detect the slot values (the parameters) from the utterance and (iv) generate a node in the local KG with an ID for this project instance. This indicates that the system starts an insert procedure on the current thread and next, it will decide about a proper action to continue the conversation towards collecting all the requested information for the project.

If the user refers an unknown entity (which does not exist from the perspective of the general KG) the system defaults. We notice that in the same utterance, the user mentions values for some parameters. The parameters are identified within the NLU module by keyword detection within the slot filling: (here code: 123, class: Python, name: TaskHelper, status: unfinished, manager: John). NLU succeeds to identify the parameters as there is an active procedure running, with an active instance ID. If there was no active instance ID, the system would default.

At turn 3, the user changes her mind and decided to remove a parameter by mentioning its value (here, remove John). The system supports two types of removal: by the name of a

Table 1. Conversation example with multiple discussion threads

Thread	**Turn**	**User utterance**	Intent	**System utterance**	Action
1	1	Hello	hello	Hello	hello
1	2	Insert a project where code is 123, class is Python, project name is TaskHelper, status is unfinished and manager is John	insert	The project will be inserted with the following parameters: code: 123, class: Python, name: TaskHelper, status: unfinished, manager: John. Is is correct?	confirmParams
1	3	Actually, I don't know yet if John could do it, please remove John	remove	The following parameters were removed: manager	removeParams
2	4	Please insert a new one	insert	I cannot insert the project unless you give me the necessary parameters: class, code, status, name; there are optional parameters too: manager	requireParams
2	5	code is 123, class is Python, name is TaskHelper, status is unfinished and manager is Marc	insert	The project will be inserted with the following parameters: code: 123, class: Python, name: TaskHelper, status: unfinished, manager is Marc. Is it correct?	confirmParams
2	6	yes	agree	You want to put a new project, but some parameters (manager) require pre-existing instances in the graph. Please insert the corresponding instances (Employee) and then proceed with the project	reject
3	7	then insert an employee with the name Marc	insert	The employee will be inserted with the following parameters: name: Marc. Is it correct?	confirmParams
3	8	yes	agree	I did insert the employee. I switched to the entity Project20230428121923835. The parameters for it are: code: 123, class: Python, name: TaskHelper, status: unfinished. If you want, you can switch to another entity from the list: Project2023028121914048	confirm
1	9	switch to Project2023028121914048	switchEntity	The swithc to Project2023028121914048 was done successfully. The parameters for it are: code: 123, class: python, name: TaskHelper, status: unfinished. If you want, you could switch to another entity from the list: Project20230428121923835	SwitchEntity
1	10	Insert a project with code 123, class is Python, name is TaskHelper, status is unfinished, and manager is John	insert	The project will be inserted with the following parameters: code: 123, class: Python, name: TaskHelper, status: unfinished, manager: John. Is it correct?	confirmParams
1	11	Manager is Marc	insert	The project will be inserted with the following parameters: code: 123, class: Python, name: TaskHelper, status: unfinished, manager: Marc. Is it correct?	confirmParams
1	12	true	agree	It seems that some values you requested have matching entities in my knowledge base: :hasManager: { ID:Employee20230428122003618; hasName:Marc;}; hasStatus:{ID:Status2; hasName: unfinished;};	reject
1	13	manager is Employee20230428122003618 and status is Status2	insert	I need you to confirm that you want to insert a project with the following parameters: code: 123, class: Python, name: TaskHelper, status: Status2, manager: Employee20230428122003618	confirmParams
1	14	yes	agree	the project was inserted succesfully. The switch to Project20230428121923835 was done successfully. The parameters for it are: code: 123, class: Python, name: TaskHelper, status: unfinished	confirm
2	15	cancel it	cancel	The ongoing procedure was stopped	cancelProcedure
2	16	thanks	thanks	for nothing	welcome
2	17	bye	goodbye	have a nice day!	reject

parameter or by its value. If the user mentions a wrong name or value, the system defaults.

At turn 4, the user decides to initiate a new procedure by saying "Please insert a new one". The system does its routine check and discovers that no new known concept was mentioned, which triggers the creation of a procedure with the old mentioned type (which is stored locally). The active ID is placed in the LIFO list and a new instance ID is generated and set as active, which starts the second thread. The old procedure is stored at the top of the LIFO list.

In turns 5-6, the conversation continues towards solving the active procedure (inserting a project instance). At turn 6, the system warns the user that she referred to an Employee instance which does not exist in the graph (she wanted to assign Marc as the manager, but there is no Marc in the graph). Therefore, the user needs to insert an employee called Marc, and only after then proceed with the project insertion.

In turn 7, the user complies with the system request and initiates a third procedure. Eventually, the third thread is created, where the system will place the current active ID in the predecessor list and generate a new procedure according to the new type given by the user (here, Employee). As the user supplied all the mandatory parameters (here, only the name), the system directly asks for confirmation of insertion.

In turn 8 the user confirms the correctness of the Employee instance and the system successfully inserts it in the general KB, which also ends the third thread. In the same time, it checks for instances in the predecessor list. If there is any, the system pops the last one inserted (LIFO approach) and gets the stored state of it from the local graph, in order to show it to the user. This switches the conversation to the second thread. Also, it prompts that the user can switch to another active instance, if there is any.

In turn 9, the user decides to make the switch to the first inserted instance (first thread). The system places the current active ID back in the predecessor list and sets the mentioned ID as active. It similarly loads its state as in turn 8.

After a few turns where the current instance is discussed, in turn 12 the system prompts the user that it found instances matching her filters in the general KG and it requests her to choose the correct IDs for each parameter. When a parameter needs to be linked to a pre-existing instance in the general KG, the system always checks its existence in two ways: by ID or by parameter value. By ID means that the user specifies an ID for the parameter and the system can directly verify its existence. Since this is not an often case, it allows checking that some instances already exists in the graph by searching for parameters' values. To do so, the system gets the mentioned value (here, Marc and unfinished) and checks each parameter of every instance of the appropriate type. If values are found, they are prompt to the user.

Turns 13-14 construct the final form of the project instance. After successfully inserting the instance in the general KG, the system switches to the last active instance from the predecessor list. This sets the second thread active again. In turn 15, the user cancels it, which triggers the removal of all associated information from the local graph. If there are no instances remaining in the predecessor list, the system does not show the switch option.

Turn 15-17 conclude the conversation. The system successfully managed three active threads, with two insertions and a canceled procedure.

We emphasize the usage of the LIFO-like list when processing the utterances of the user in turns 4 and 7, which determine the system to create novel discussion threads. In the response of turn 8, together with confirming the insertion of the employee in the general KG, the system proposes the user to continue with the previously procedure on a previously active instance. This represents a natural way to move along the discussion threads, without imposing the user some pre-established processing order.

The ontology is a blueprint to bootstrap the knowledge of the system. It has several concepts and relationships, while cardinality defines their compulsoriness. A larger ontology may fit the system if it respects its current rules; each relationship that emulates a parameter needs to follow the template ‘hasAttribute’ for its name. Future work will improve the compatibility with larger ontologies, but only after we succeed integrating a DL component into the NLU module, to replace rule-based intent detection with neural network models. This leads to the possibility of removing built-in rules for both the ontology and the system, to increase compatibility. Also, we plan on migrating to a KG server like GraphDB[4], to enable advanced and scalable reasoning.

The current version of the system does not allow concept learning (i.e. to store novel concepts in the global KG, rather than instances), but ideas were drawn in this direction, as KGs provide plenty of opportunities. One idea is to integrate a module that enables the user to describe a desired concept that is to be added to the ontology. In this way, the system can naturally grow its knowledge through direct conversations with a user. One drawback is raised by the architecture of the ontology. If it is too complex, the user might not understand what to do, leading to more fined-tuned rules for the system to deal with such situations. Another idea is to introduce Reinforcement Learning models in the system’s architecture. Far more powerful than simple rules, such models can learn paths and behaviors that were not thought of by the system’s architect. The disadvantage is the lack of training data and the complexity of the learning process of such models.

One of the main advantages of our system is that we can generate annotated datasets for training various NLP tasks which are related to TOD systems. For example, within the NLU module, we annotate data for NER (Named Entity Recognition) task by labeling each relevant detected slot with an entity type in the IOB2[11] format. For intent and slots detection tasks, we label each phrase with the detected intent and assign a key-value pair for each detected slot.

Furthermore, we can leverage the likes of MultiWOZ[4] or Schema [12] datasets, where each conversation is labeled for all four modules (NLU, DST, POL, NLG). In fact, this is our future goal: to train different Deep Learning modules using datasets generated with the current version of the system and then integrate them with it. This enables the generation of more natural dialogues, which leads to more robust DL models.

5. Conclusion

In this paper we present the architecture of a task-oriented dialog system built with two specific purposes: to grow the knowledge base of a target company and to prepare future datasets with annotated conversations regarding the business procedures taking place within the company. Solving the above-mentioned objectives will allow us to advance towards the future purpose of our TOD system. This is to help the employees within the company to accomplish their business tasks, by leveraging expert knowledge existing within the company.

The architecture of our system is a classical one, composed of the four modules recommended by the literature: Natural Language Understanding (NLU), Dialogue State Tracking (DST), Policy management (POL) and Natural Language Generation (NLG). We use the standard approach with rules and conversation frames for understanding the user utterances and producing the system response.

We cover the Create-Retrieve-Update-Delete (CRUD) procedures for acquiring and maintaining relevant knowledge from the human user. The knowledge is persisted in a global knowledge graph. As an architectural novelty, during the conversation we grow a local knowledge graph where we acquire the knowledge temporarily learned from the user’s utterances, rather than globally managing the conversation state within the DST module. Furthermore, it permits the system to tackle multiple procedures inside just one conversation by managing multiple discussion threads, as opposed to the majority of TOD systems in the literature, where each conversation deals with a single thread or business procedure. With a running example, we demonstrate the functioning of the system.

[4]`https://graphdb.ontotext.com/`

Future work will consists of empowering the system with a complex ontology suited for some specific business processes within the organization. Based on the acquired dataset, we will be capable of constructing advanced deep learning models for solving the NLP tasks which are part of the four modules of the system. With them, our system will gain more flexibility and naturalness, becoming a business assistant within the company.

Acknowledgement

The present work received financial support through the project: *Integrated system for automating business processes using artificial intelligence*, POC/163/1/3/121075 - a Project Co-financed by the European Regional Development Fund (ERDF) through the Competitiveness Operational Programme 2014-2020.

V.I. Iga thanks Babes-Bolyai University for granting him the special scholarship for scientific activity during the academic year 2022-2023.

References

1. J. Andreas, J. Bufe, D. Burkett, C. Chen, J. Clausman, J. Crawford, K. Crim, J. DeLoach, L. Dorner, J. Eisner, H. Fang, A. Guo, D. Hall, K. Hayes, K. Hill, D. Ho, W. Iwaszuk, S. Jha, D. Klein, J. Krishnamurthy, T. Lanman, P. Liang, C. H. Lin, I. Lintsbakh, A. McGovern, A. Nisnevich, A. Pauls, D. Petters, B. Read, D. Roth, S. Roy, J. Rusak, B. Short, D. Slomin, B. Snyder, S. Striplin, Y. Su, Z. Tellman, S. Thomson, A. Vorobev, I. Witoszko, J. A. Wolfe, A. Wray, Y. Zhang, and A. Zotov. Task-oriented dialogue as dataflow synthesis. *Trans. Assoc. Comput. Linguistics*, 8:556–571, 2020.
2. A. Bordes, Y. Boureau, and J. Weston. Learning end-to-end goal-oriented dialog. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 2017*. OpenReview.net, 2017.
3. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.
4. P. Budzianowski, T. Wen, B. Tseng, I. Casanueva, S. Ultes, O. Ramadan, and M. Gasic. Multiwoz - A large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2018*, pages 5016–5026. Association for Computational Linguistics, 2018.
5. D. Chaudhuri, M. R. A. H. Rony, and J. Lehmann. Grounding dialogue systems via knowledge graph aware decoding with pre-trained transformers. In R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski, and M. Alam, editors, *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 323–339. Springer, 2021.
6. H. Chen, X. Liu, D. Yin, and J. Tang. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 19(2):25–35, 2017.
7. J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, 2019, Volume 1 (Long and Short Papers)*, pages 4171–

4186. Association for Computational Linguistics, 2019.

8. D. Jurafsky and J. Martin. *Speech and Language Processing: an Introduction, 2nd ed., 15th imp.* Pearson, 2022.
9. Z. Li, H. Wang, A. Albalak, Y. Yang, J. Qian, S. Li, and X. Yan. Making something out of nothing: Building robust task-oriented dialogue systems from scratch. In *Alexa Prize TaskBot Challenge Proceedings*, 2022.
10. A. Nazir, M. Y. Khan, T. Ahmed, S. I. Jami, and S. Wasi. A novel approach for ontology-driven information retrieving chatbot for fashion brands. *International Journal of Advanced Computer Science and Applications*, 10(9), 2019.
11. L. A. Ramshaw and M. Marcus. Text chunking using transformation-based learning. In D. Yarowsky and K. Church, editors, *Third Workshop on Very Large Corpora, VLC@ACL 1995, Cambridge, Massachusetts, USA, 1995*, 1995.
12. A. Rastogi, X. Zang, S. Sunkara, R. Gupta, and P. Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, 2020*, pages 8689–8696. AAAI Press, 2020.
13. Y. Su, L. Shu, E. Mansimov, A. Gupta, D. Cai, Y. Lai, and Y. Zhang. Multi-task pre-training for plug-and-play task-oriented dialogue system. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 4661–4676. Association for Computational Linguistics, 2022.
14. Y. Tuan, S. Beygi, M. Fazel-Zarandi, Q. Gao, A. Cervone, and W. Y. Wang. Towards large-scale interpretable knowledge graph reasoning for dialogue systems. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, 2022*, pages 383–395. Association for Computational Linguistics, 2022.
15. T. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P. Su, S. Ultes, and S. J. Young. A network-based end-to-end trainable task-oriented dialogue system. In M. Lapata, P. Blunsom, and A. Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, 2017, Volume 1: Long Papers*, pages 438–449. Association for Computational Linguistics, 2017.
16. J. D. Williams, A. Raux, and M. Henderson. The dialog state tracking challenge series: A review. *Dialogue Discourse*, 7(3):4–33, 2016.
17. C. Wu, S. C. H. Hoi, R. Socher, and C. Xiong. TOD-BERT: pre-trained natural language understanding for task-oriented dialogue. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, 2020*, pages 917–929. Association for Computational Linguistics, 2020.
18. S. Yang, R. Zhang, and S. M. Erfani. Graphdialog: Integrating graph knowledge into end-to-end task-oriented dialogue systems. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, 2020*, pages 1878–1888. Association for Computational Linguistics, 2020.