Conversational Agent in Romanian for Storing User Information in a Knowledge Graph

Gabriel Boroghina, Dragos Georgian Corlatescu, Mihai Dascalu
University Politehnica of Bucharest
313 Splaiul Independenței, Bucharest, Romania
gabrielboroghina@outlook.com, {dragos.corlatescu, mihai.dascalu}@upb.ro

DOI: 10.37789/rochi.2020.1.1.15

ABSTRACT

Information surrounds us and keeping track of relevant details can be challenging. Although there are multiple applications to take notes, organize ideas, or set reminders, existing solutions are semantic-agnostic and rely on the user to manually search for desired information by keywords. We propose a novel method to help people store and retrieve such details with ease in Romanian language. Our conversational agent built on top of the RASA framework is capable to extract relevant information from the user's utterances, store them in a persistent knowledge graph, and ultimately, access them when requested. A set of specific intents regarding locations, timestamps, and properties were created and learned by the agent using manually built examples. In addition, an interaction logic based on a knowledge graph was added to enable the storage and retrieval of information, based on the identified semantic components from the input sentences. The performed tests showed a good accuracy for intent detection, and promising results for the sentence parser. Our conversational agent is accessible as a web application which can process text or speech inputs, and responds with a textual representation of the user's memorized facts.

Author Keywords

Natural Language Processing; Conversational agent; Knowledge representation.

ACM Classification Keywords

H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

I.2.7 Natural Language Processing: Discourse, Language parsing and understanding, Text analysis

General Terms

Natural Language; Text analysis.

INTRODUCTION

Conversational agents represent intelligent programs based on Natural Language Processing (NLP) techniques, capable of performing dialogue conversations with a human interlocutor. These agents need to maintain the global state of the conversation and provide responses in relation to this state and the user's messages. Moreover, conversational agents became more and more adopted by the industry, mostly in the form of chatbots that offer guidance to clients with regards to a product or service, but also as virtual assistants capable to perform various tasks at the user request, such as taking notes, setting an appointment, and even controlling smart home devices.

Conversational agents are preferred in a wide range of cases due to their programmability and their scalability for simple and highly repeatable tasks, such as basic technical support. Equally important is the help agents can provide to people who are unfamiliar with the use of a technology, or have deficiencies (such as blindness), by allowing them to simply talk with the agent in order to execute specific tasks. Thus, conversational agents try to push the perceived intelligence of computers and their ability to interact with people further, at an increased pace.

We introduce a memory-assistant conversational agent for Romanian language, capable to remember the user's previously registered information. The agent is capable of processing the user's statements that expose information which should be memorized, and can also react to his/her requests for a specific detail, relying upon the gathered data. The core features consist in detecting the users' intents (whether they wanted to register or to access a specific type of information), and extracting the key facts from their utterances, storing them in a knowledge graph, and exposing them back on demand.

Our agent works with factoid information [15], meaning that it has the ability to detect specific elements inside more complex sentences, and respond to queries such as: "where" (locations of objects or events), "when" (timestamps of events or actions), "which is" (various properties of entities), "which of the" (the instance of an entity that has a certain qualification or function), or "who" (subject of an action or a description). The agent operates in a user-initiative mode, meaning that the user launches the dialogue sessions with a request, waiting for the agent to reply with the corresponding result. Moreover, the information is presented in a concise form to the user, just as it is stored in the knowledge graph; no context planning or natural language generation techniques are used for information retrieval. Users do not have to specify the type of the information to be stored, nor how or where to keep it. The chatbot itself is responsible for interpreting, understanding, and managing data, while the interaction with the user is limited to expressing a fact or a request in natural language (text or speech). In this manner, the agent allows for fast, natural, and intuitive information registration and retrieval.

STATE OF THE ART

Conversational agents are increasingly adopted into different software applications and web platforms. According to Brandtzaeg and Følstad [6], the main benefits of using such agents are their productivity (i.e., obtaining support or necessary information regarding a product or a service), but also entertainment, or social interaction and communication (e.g., chit chat or social chatbots). The bots provide a significantly enhanced user experience and interactivity for performing various tasks than a static user interface. In addition, bots also offer scalability and objectivity in their conversation with the user. Thus, a conversational agent can be perceived as being more agile than a person at providing simple information or helping the user solve standard problems that fit a certain template.

Depending on the complexity of the conversation, two types of agents come into view, according to Jurafsky and Martin [15]: a) simple agents designed to perform tasks at the user's request, called task-oriented dialogue agents, and b) chatbots, which are more complex agents that can maintain longer conversations, which can extend over several replies and can include different interleaved subjects of discussion. In this article, we will focus on the first category.

Conversational agents rely on a dialogue-state architecture [15] that addresses two problems: intent classification and slot filling. Multiple recent approaches try to develop joint models capable to perform both tasks on a single pass through the model. For instance, Liu and Lane [19] proposed an attention-based RNN (Recurrent Neural Network) model which can concurrently perform intent detection and slot filling, for a given token sequence. The model consists of a bidirectional RNN, based on LSTM (Long Short-Term Memory) cells. The model generates slot labels for each token based on a hidden state made up of a forward and a backward state obtained by analyzing the input sequence in both directions. Attention provides additional information about the input sequence through a context vector that is combined with the hidden state before token labelling. Intent detection is performed simultaneously by using the computed hidden states from the bidirectional RNN model. An attention-based encoder-decoder model was also investigated by Liu and Lane [19]; the model integrates all information from the input sequence (encoding), and then generates an output sequence containing the slot labels (decoding).

In contrast to the previous approach, Wang et al. [28] contradict the joint model structure, and propose a bi-model RNN structure, consisting in two connected BiLSTM components, one for intent detection and one for slot filling, that takes advantage of the cross-impact between the two tasks. The two BiLSTM models exchange their hidden states and combine them when generating the output of each task; thus, the intent detection model benefits from the features extracted by the slot filling model and vice versa. A slight variation containing one LSTM decoder on top of each BiLSTM component was also explored, obtaining even better results and surpassing the previous state of the art systems for both tasks.

Dialogue-state architecture can be also used for designing task-oriented dialogue systems [15]. These architectures consist of a more complex pipeline of processing components [15]: automatic speech recognizer, spoken language understanding unit, conversation state tracker, dialogue policy, natural language generation unit, and speech synthesizer. While the first two components act as blackboxes that perform processing tasks over the user's voice input and export texts, the dialogue tracker is a stateful component that maintains at each moment the conversation status, alongside the chat history.

Despite the usefulness and remarkable user experience conversational agents can provide, their creation from scratch can prove to be challenging and time consuming. An open-source framework that comes in handy is RASA [5], which provides Natural Language Understanding (RASA NLU) and Dialogue Management (Rasa Core) features for building intelligent conversational agents with a minimum effort from the programming point of view. The framework is based on a dialogue-state architecture, but supports only the communication through text messages (i.e., it does not integrate speech-to-text and text-to-speech converters). The NLU component has a modular pipeline, instrumental in customizing and tweaking the components involved in processing and interpreting the user's utterances. Tracker objects are used to manage the conversation flow; their role is to maintain a list of events encountered during the communication session, together with relevant facts (slots) exposed by users through their statements.

RASA relies on the DIET (Dual Intent and Entity Transformer) classifier [8] for intent detection. The model obtains a good performance and its training is significantly faster and easier when compared to older models. The architecture can be visualized in Figure 1. The model uses a 2-layer Transformer [27] for condensing the semantic features extracted from each token, and compares the resulting feature vector with the vector corresponding to the gold intent based on a similarity (dot product) metric. A Conditional Random Field (CRF) [17] can be used on top for entity extraction. Pre-trained word vectors from BERT [12] or GloVe [21] can be used to improve the results.

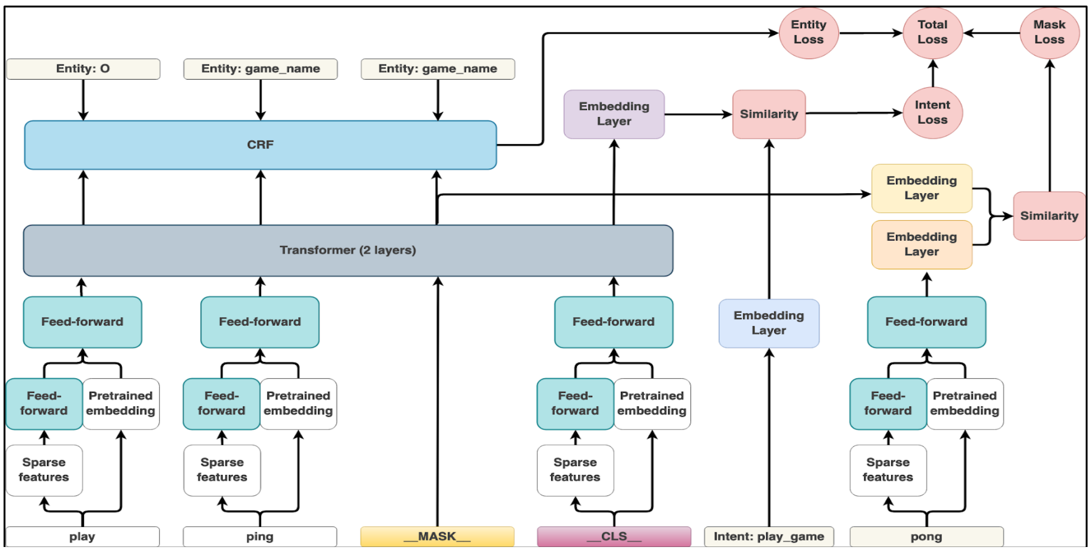


Figure 1. DIET architecture [8].

Data storage is also a central component for our agent. A traditional SQL approach is not suitable because data from the conversation is unstructured and in natural language. Thus, semantic networks [26] (i.e., directed graph where nodes represent entities or concepts, and the arcs mark relationships between them) are preferred. Semantic networks provide a better contextualization in terms of the representation and identification of knowledge facts. Knowledge graphs are semantic networks which include inference methods for deriving and combining information in order to extract new facts. NoSQL graph databases include different query languages to read or modify stored data [1]. The most relevant examples of such query languages are Cypher [13] (which introduces powerful pattern-matching based queries), SPARQL [23] (designed for RDF data storage schemas), GraphQL (a data query language for APIs) and Gremlin [25] (which displays a distinguishable functional style in its queries).

Different approaches for structuring information in graph databases were considered for representing knowledge in our conversational agent [22]. One of them, the labeled-property graph model (used, for example, by the Neo4j Database Management System [20]) consists of adding properties to nodes or relationships to specify different metadata in a key-value fashion. This approach enables a flexible and compact representation of information. Conversely, the RDF model [18] allows nodes and arcs only to specify an URI label, which in return defines the class of the entity or a relationship. This leads to a more strict and uniform data representation, but, at the same time, to a reduced topological conciseness. Additional properties have to be injected as separate nodes due to the simplicity of the RDF [18] graph elements.

METHOD

Corpus

Two supervised learning models are involved in the Natural Language Understanding pipeline of our agent: the first for detecting the intent from the user's utterances, and the second for parsing the sentences. These models implied the creation of two datasets containing labeled examples.

Intent Classification

The classification of the user's sentences by their scope requires defining a set of intents that determines the general types of tasks handled by the agent. These intents outline the constrained world or domain the agent was designed for. The considered intent classes are presented in Table 1.

The training dataset consists of 285 manually built sentences (for the "get" and "store" intents) and phrases (for auxiliary intents, such as greetings or the store request announcement). Additionally, a separate test set comprising of 128 examples was created to evaluate the performance of the intent detector. The distribution of examples in the training and test datasets can be observed in Figure 2.

Syntactic Parsing

Due to lack of a specific dataset for our task in Romanian, the training dataset consists of 200 manually annotated examples, while the test set used to measure parsing accuracy contains another 70 sentences. The examples were created using a custom-made HTML and JavaScript web interface that led to a faster visual annotation process. **Error! Reference source not found.**Figure 3 shows the frequencies for each syntactic question introduced in the training set. An example with an annotated sentence can be observed in Figure 4. The head represents the index of the word in the sentence to which another word is connected to in the dependency tree; the „deps" property includes custom word tags used later on.

Table 1. Intents description.

Intent	Description
Greet, Goodbye	Can be used at the beginning or the end of an interaction with the agent, although their use is optional. General greeting phrases were used for these intents, as training examples.
Store request	The user announces the agent that they want to store a piece of information in the knowledge graph.
Store/Get location	The user wants to tell/ask the agent about the location of an entity or an event. For location requests, the agent expects a sentence consisting of the interrogative pronoun "unde?" (eng., "where?"), along with a subject entity or an action involving a direct object. To store a location, a sentence containing a place adverbial and either a subject (e.g., tell where the subject is placed) or a direct object (e.g., tell where the user puts an object) is expected by the agent.
Store/Get timestamp	The sentence tells/asks the agent about the timestamp (time point, duration, or frequency) of an event. The phrases "când?" (eng., "when?"), "de când?" (eng., "since when?"), "până când?" (eng., "until when?") and "cât timp?" (eng., "how long?") are used to ask the agent for this type of information, along with an event expressed as a bare noun phrase or as a complex action (containing also other semantic entities such as adverbials or direct objects). A sentence containing a time adverbial is expected to register a time information.
Store/Get attribute	The user wants to store or retrieve the value of an entity attribute (a personal detail of that entity, which can range from a person's phone number to the dimensions or the price of an object). The "get attribute" intent is represented, in the agent's view, through a question involving a phrase of the type "care (a fi)?" (eng., "what is?"), followed by a noun phrase describing the requested property (and optionally its owner). To store an attribute, a sentence containing a subject (the entity) and a predicate (specifying the attribute value) is expected by the agent.
Store following attribute	Similar to the previous action, this intent allows the user to store an attribute, but in two steps: the first utterance is matched to this intent category and contains the attribute name, whilst the second utterance includes the actual attribute value. This approach enables users to store complex values that should be taken as a whole, instead of being parsed into component tokens.
Get subject	The user asks for the entity that represents the subject of an action or has a specific property according to the knowledge base. This intent is triggered through the interrogative pronoun "cine?" (eng., "who?"), alongside an action or a state.

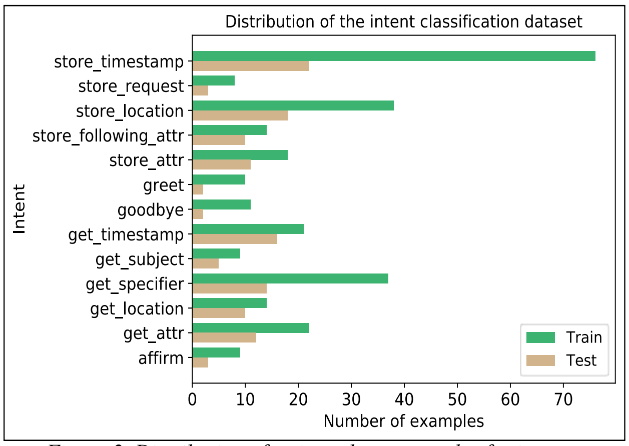


Figure 2. Distribution of train and test examples for intent classification.

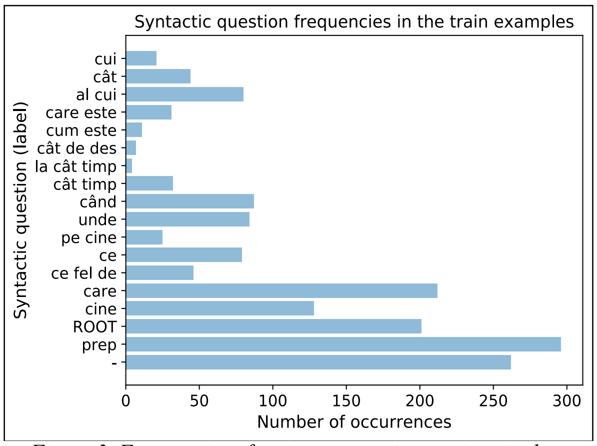


Figure 3. Frequencies of various syntactic questions in the training examples.

```
{
    "sentence": "permisul meu de conducere e în torpedoul de la mașină",
    "parsing": {
        "heads": [4, 0, 3, 0, 4, 6, 4, 8, 9, 6],
        "deps": ["cine", "al cui", "prep", "care", "ROOT", "prep", "unde", "-", "prep", "care"]
    }
}
```

Figure 4. Annotated sentence example.

Application Architecture

The application is divided into several components that perform sequential processing tasks, starting from the user's utterances (speech or text), through the NLU (Natural Language Understanding) pipeline, to the knowledge base, and back to the user with responses to their information requests, as depicted in Figure 5.

Processing Pipeline

Our conversation agent is built on top of the RASA framework for conversational agents [5], which provides two chained pipelines: the first for NLU, while the second (RASA Core) is centered on dialogue management (i.e., selecting the proper response to the user's input and advancing the conversational flow).

The modularity of the framework allows the customization of the pipeline components; thus, an application-specific processing pipeline was designed. The first components are responsible for tokenization and feature generation. They are based on the spaCy [14] Romanian model (available in the open-source ReaderBench framework [10]), which is trained on the Romanian Universal Dependencies treebank [4]. This approach considers pretrained word embeddings, which offer additional linguistic features that support the next NLU components (such as the intent classifier) to obtain better performance. In addition to the previous components, other featurizers predefined in RASA, namely the Lexical-Syntactic Featurizer and the Count Vectors Featurizer were added. The latter was configured to create the bag-of-words representation only at token level because the character and the n-gram levels were generating too much variance in the results. Finally, a custom component for syntactic parsing was integrated in the NLU pipeline, based on a pretrained model and a set of custom actions provided to the RASA Core pipeline to be executed (depending on the response selection result) as a reply to the user's request. These actions are responsible for establishing a connection to the database, as well as returning the results to the user or performing the database update.

User Interface

The User Interface (UI) of our conversational agent consists of a single-page web application (HTML, CSS and JavaScript) based on the React framework [24] (see Figure 7). The interface consists of a chat window in which the exchange of messages conducted with the agent in the current session are displayed. The UI has a multimodal input as the user can interact with the bot either by typing the requests in an input text field from the chat window, or by uttering a statement after launching the speech input mode.

Speech-to-text conversion is performed through the Web Speech API [9]. It is a JavaScript API, currently specified as a W3C draft, independent of the underlying algorithm implementation that provides a unified interface for performing both automatic speech recognition and speech synthesis in the context of web applications.

Communication with the core agent is based on the RASA's HTTP REST API, which is used to deliver the user's messages to the bot, and then wait for its response. The requests are executed asynchronously using the Axios HTTP client [3], which ensures a fluid interaction between the user and the agent.

The complete communication between the modules of the application can be observed in Figure 6. Note that, although custom actions are also part of the RASA framework, these services run as a separate server which is accessed from the core agent through an HTTP endpoint.

The application follows a modular MVC (Model-View-Controller) architecture [16], where the view is represented by the web interface, the custom actions together with the Neo4j database define the data model, whereas the core RASA agent acts as a controller. The RASA agent receives the user's requests from the view, processes them, and sends the results to the model component, which manages data representation. Conversely, the controller builds the response for the user based on the extracted data and delivers it by means of the frontend view, which presents it to the user.

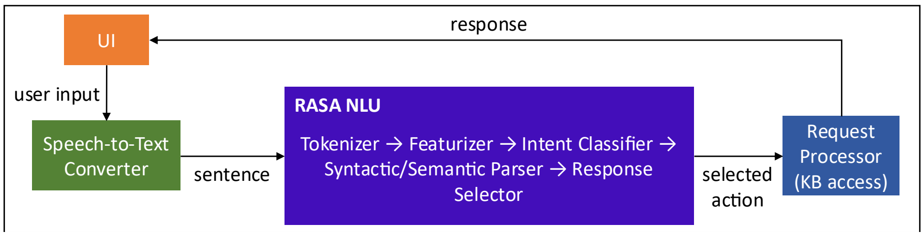

response
UI
user input
Speech-to-Text Converter
sentence
RASA NLU
Tokenizer → Featurizer → Intent Classifier → Syntactic/Semantic Parser → Response Selector
selected action
Request Processor (KB access)

Figure 5. Logical flow diagram of the application.

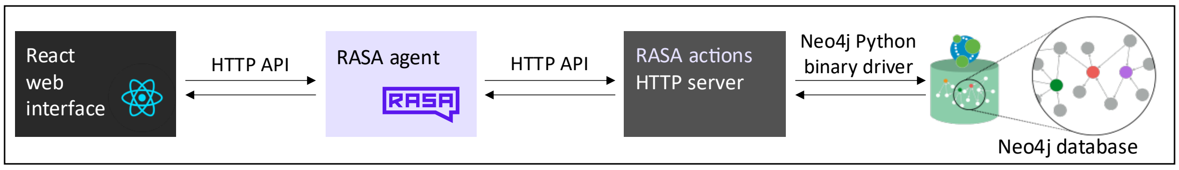
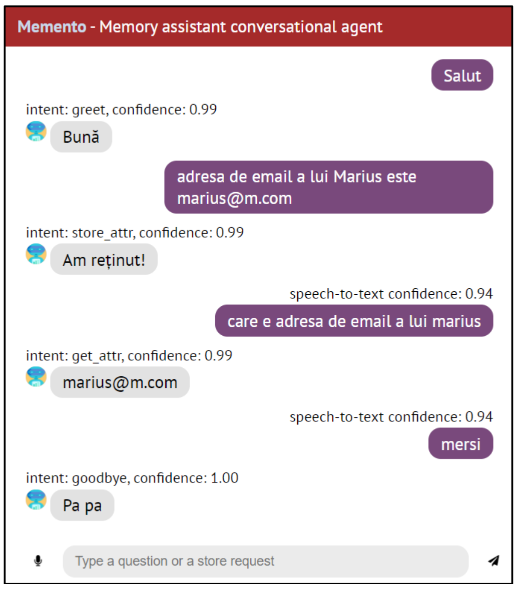


Figure 6. Main application modules.


Memento - Memory assistant conversational agent

Salut

intent: greet, confidence: 0.99

Bună

adresa de email a lui Marius este marius@m.com

intent: store_attr, confidence: 0.99

Am reținut!

speech-to-text confidence: 0.94

care e adresa de email a lui marius

intent: get_attr, confidence: 0.99

marius@m.com

speech-to-text confidence: 0.94

mersi

intent: goodbye, confidence: 1.00

Pa pa

Figure 7. Application interface.

Intent Classifier

Our agent maps the user sentences to a limited set of predefined intents, in accordance with Table 1. The intent classifier allows the request processor to perform a more customized entity selection, depending on the relevant information included in the input phrase. In addition, the classifier gives the agent the ability to execute an action specific to an identified intent, i.e. run a query or update the knowledge graph. A noteworthy aspect is that a single intent may be selected for an utterance, meaning that no more than one kind of relevant information (specific to that intent) is identified and considered by the agent.

Intent classification is performed using RASA's DIET classifier. Corresponding part of speech tags for each word were provided to the classifier by customizing RASA's Lexical-Syntactic Featurizer in order to accommodate the particularities of statements specific to each intent. Additionally, the RASA entities occurring in the examples of each type of intent were declared. The presence or the absence of such entities in an example can be used, according to the RASA documentation, as features by the DIET classifier to improve its accuracy.

Request Processor

After the intent from the user statement is predicted and the semantic entities are extracted, the processing pipeline continues by executing an action specific to a given intent. The action is selected based on the stories provided as training data to the RASA Core pipeline. The stories consist in general of chains of pairs (intent, reply action), describing possible conversational flows. However, since the majority of our intents imply only a request-response exchange, the associated stories only include the mapping between the intent and the custom action that accesses the knowledge base to process the user's inquiry. Additionally, an acknowledgement message follows when storing new information to confirm the action's fulfillment, so that the user knows the information was registered. A slightly different case is the one of the "store following attribute" intent because the attribute name needs to be retained between two replies. This is achieved by means of a RASA form, which is initiated once the first user's message (the store request) is uttered. At that point, a slot is filled with the attribute description. The form is completed after the user inserts the value of the attribute as the second reply. Afterwards, the action responsible for storing the attribute is executed.

Knowledge Representation

Our knowledge base is based on the Neo4j graph database management system, which ensures a powerful semantic representation of the entities and the facts associated to them. A notable benefit is the lack of need for an explicit database schema, meaning that each entity may have its own types of attributes or relationships with other entities. This facilitates the storage of heterogeneous information transmitted by the user. Another reason for choosing Neo4j is Cypher, its query language [13], which enables pattern matching queries. Hence topological structures such as nodes, relationships, paths or subgraphs can be matched using a pattern that intrinsically specifies labels or properties of the nodes and the relations that link them. The reason for selecting Neo4j resides in its ease of integration with RASA. Currently, there are 4 alternative solutions that can be used [7]. Our differentiating criterion between them consisted of publicly available performance benchmarks [11].

Entities defined as noun phrases are represented as a tree in the graph, where each node identifies a syntactic component of the noun phrase. The decomposition is performed in the following manner:

- The root (noun) of the noun phrase constitutes the base class of the entity.
- Each span with the syntactic function of the attribute, responding to one of the questions "care?" (eng., "which?") or "ce fel de?" (eng., "what kind of?") represents a "specifier", having the role of identifying a particular instance of the base class.
- The token responding to the question "al cui?" (eng., "whom") identifies the owner of the entity, and therefore it is placed as the root of the entity's tree representation. This token can be also classified as a "specifier".

RESULTS

The customized intent classifier from RASA exhibited very good results at identifying the intents from the test sentences, as shown in the confusion matrix from Figure 8 where all test inputs were correctly classified. Confidence was around 95% on all entries from the test dataset, denoting a strong differentiation between intents.

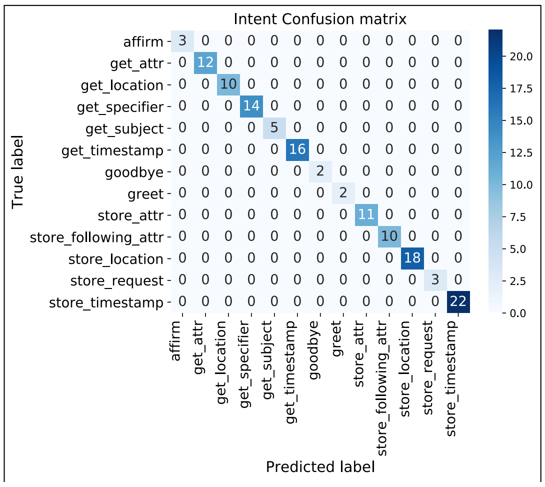


Figure 8. Confusion matrix for the intent classification task.

The accuracy of the syntactic parser is high (93% for dependency heads). Figure 9 depicts the confusion matrix resulted after running the parser on our test dataset consisting of 70 examples. Syntactic dependencies can occur in different structures and positions within a sentence; thus, a larger variety in the training dataset is required to ensure the model's capability to generalize. However, our current set of examples, which were manually created and annotated, is quite limited and needs to be extended.

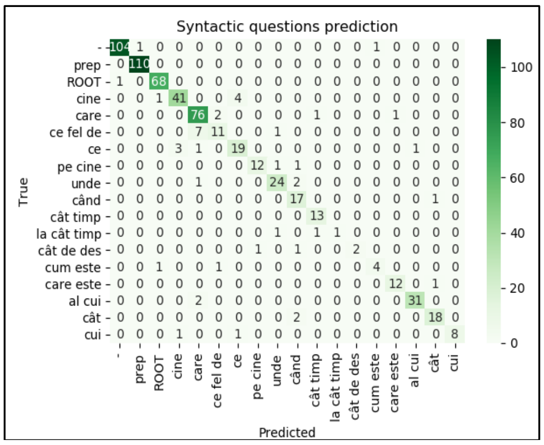


Figure 9. Confusion matrix for the syntactic parsing.

Problems also occurred when the input sentence included tokens not belonging to any of the available syntactic dependency classes. Nevertheless, the classifier tries to map these tokens to a class, which in many cases was not the "-" dependency (i.e., the best option in this case). This results in wrong labels, and may also lead to words around the misclassified tokens being mapped to an incorrect dependency. Table 2 includes the precision, recall, and F1 scores obtained for each type of syntactic question. Only the labels' attachment is considered, without taking into account the placement of the dependency heads.

Table 2. Syntactic questions classification.

Type	P	R	F1-score	#
-	0.99	0.98	0 .99	106
ROOT	0.97	0.99	0.98	69
al cui (whose)	0.97	0.94	0.95	33
care (which)	0.87	0.95	0.91	80
care este (which is)	0.92	0.92	0.92	13
ce (what)	0.79	0.79	0.79	24
ce fel de (what kind of)	0.79	0.58	0.67	19
cine (who)	0.91	0.89	0.90	46
cui (to whom)	1.00	0.80	0.89	10
cum este (how is it)	0.80	0.67	0.73	6
când (when)	0.74	0.94	0.83	18
cât (how much)	0.90	0.90	0.90	20
cât de des (how often)	1.00	0.50	0.67	4
cât timp (how much time)	0.87	1.00	0.93	13
la cât timp (how long)	1.00	0.33	0.50	3
pe cine (who)	0.92	0.86	0.89	14
unde (where)	0.89	0.89	0.89	27
preposition	0.99	1.00	1.00	110
Accuracy			0.93	615
Macro avg	0.91	0.83	0.85	615
Weighted avg	0.93	0.93	0.93	615

CONCLUSIONS

In this article we introduced a chatbot for Romanian capable of storing and retrieving information from and to users. Several components were designed, namely a web interface, an intent classifier, a syntactic parser, custom actions dedicated to each intent, and a graph database manager. The user interface consists of a React web page that mediates the communication between the client and the conversational agent. Speech-to-text capabilities were added to facilitate the user interaction by using the Web Speech API.

As for future improvements, we consider a smarter data matching algorithm to identify the requested information, even if the request is not completely similar to the stored information (for example, words replaced with synonyms, or missing prepositions linking tokens from a noun phrase). In addition, we want to integrate additional information from general knowledge graphs (e.g., DBPedia [2]). This would allow the agent to match entities, states, or actions in a more generalized manner, resulting in a more intelligent and helpful behavior. Another future step to ensure increased performance and robustness would consist of enhancing the syntactic parser component. We strive to handle any type of statements, including those containing subordinate sentences, with all potential types of syntactic dependencies.

ACKNOWLEDGMENT

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-III 72PCCDI / 2018, ROBIN – "Roboții și Societatea: Sisteme Cognitive pentru Roboți Personali și Vehicule Autonome".

REFERENCES

1. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., and Vrgoč, D., 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR) 50*, 5, 1–40.
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z., 2007. DBPedia: A nucleus for a web of open data. In *The semantic web* Springer, 722–735.
3. Axios, n.d. Axios homepage. Retrieved July 27th 2020 from https://github.com/axios/axios.
4. Barbu Mititelu, V., Ion, R., Simionescu, R., Irimia, E., and Perez, C.-A., 2016. The Romanian Treebank Annotated According to Universal Dependencies. In Proceedings of the 10th Int. Conf. on Natural Language Processing (HrTAL2016) (Dubrovnik, Croatia).
5. Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A., 2017. Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*.
6. Brandtzaeg, P.B. and Følstad, A., 2017. Why people use chatbots. In Proceedings of the International Conference on Internet Science (Thessaloniki, Greece), Springer, 377–392.
7. Bunk, T., 2019. Set up a knowledge base to encode domain knowledge for Rasa. Retrieved September 27th 2020 from https://blog.rasa.com/set-up-a-knowledge-base-to-encode-domain-knowledge-for-rasa/.
8. Bunk, T., Varshneya, D., Vlasov, V., and Nichol, A., 2020. DIET: Lightweight Language Understanding for Dialogue Systems. *arXiv preprint arXiv:2004.09936*.
9. Community Group Report, 2020. Web Speech API. Retrieved July 27th 2020 from https://wicg.github.io/speech-api/.
10. Dascalu, M., Dessus, P., Bianco, M., Trausan-Matu, S., and Nardy, A., 2014. Mining texts, learner productions and strategies with ReaderBench. In *Educational Data Mining: Applications and Trends*, A. Peña-Ayala Ed. Springer, Cham, Switzerland, 345–377.
11. DB-Engines, 2020. DB-Engines Ranking of Graph DBMS. Retrieved September 27th 2020 from https://db-engines.com/en/ranking/graph+dbms.
12. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
13. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., and Taylor, A., 2018. Cypher: An evolving query language for property graphs. In Proceedings of the 2018 Int. Conf. on Management of Data, 1433-1445.
14. Honnibal, M. and Montani, I., 2017. spacy 2: Natural language understanding with bloom embeddings. *convolutional neural networks and incremental parsing 7*, 1.
15. Jurafsky, D. and Martin, J.H., 2009. *An introduction to Natural Language Processing. Computational linguistics, and speech recognition*. Pearson Prentice Hall, London.
16. Krasner, G.E. and Pope, S.T., 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming 1*, 3, 26–49.
17. Lafferty, J., McCallum, A., and Pereira, F.C., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the 18th Int. Conf. on Machine Learning 2001 (ICML 2001) (Williamstown, MA, USA), ACM, 282–289.
18. Lassila, O. and Swick, R.R., 1998. *Resource description framework (RDF) model and syntax specification.* World Wide Web Consortium.
19. Liu, B. and Lane, I., 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*.
20. Miller, J.J., 2013. Graph database applications and concepts with Neo4j. In Proceedings of the Southern Association for Information Systems Conference (Atlanta, GA, USA).
21. Pennington, J., Socher, R., and Manning, C.D., 2014. Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods on Natural Language Processing (EMNLP 2014) (Doha, Qatar), ACL.
22. Pokorný, J., 2015. Graph databases: their power and limitations. In Proceedings of the IFIP Int. Conf. on Computer Information Systems and Industrial ManagementSpringer, 58–69.
23. Prud'hommeaux, E. and Seaborne, A., 2017. SPARQL query language for RDF. W3C Recommendation (2008) World Wide Web Consortium.
24. React, n.d. React website. Retrieved July 27th 2020 from https://reactjs.org/.
25. Rodriguez, M.A., 2015. The Gremlin graph traversal machine and language (invited talk). In Proceedings of the 15th Symposium on Database Programming Languages (Pittsburgh, PA, USA), 1–10.
26. Sowa, J.F., 2014. Principles of semantic networks: Explorations in the representation of knowledge Morgan Kaufmann, San Mateo, CA, USA.
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., and Polosukhin, I., 2017. Attention is all you need. In Proceedings of the 31st Conf. on Neural Information Processing Systems (NIPS 2017) (Long Beach, CA, USA), 5998–6008.
28. Wang, Y., Shen, Y., and Jin, H., 2018. A bi-model based RNN semantic frame parsing model for intent detection and slot filling. *arXiv preprint arXiv:1812.10235*.