A Dialog Manager for Micro-Worlds

Radu ION, Valentin Gabriel BADEA, George CIOROIU, Verginica BARBU MITITELU, Elena IRIMIA, Maria MITROFAN, Dan TUFIȘ*

Romanian Academy Research Institute for Artificial Intelligence "Mihai Drăgănescu",
13 September 13 Street, Bucharest, 050711, Romania
radu@racai.ro, valentin.gabriel.badea@gmail.com, gcioroiu@racai.ro, vergi@racai.ro,
elena@racai.ro, maria@racai.ro, tufis@racai.ro (**Corresponding author*)

Abstract: The paper describes the micro-world-based dialog manager which was developed in the ROBIN project. The manager was designed to be loaded into the Pepper robot, used in real-world scenarios and interface with real-time automatic speech recognition and synthesis for Romanian language. A strict requirement for the development of the dialog manager was that it had to be configurable, with minimum user intervention, to a wide range of dialogue situations, such as assisting the elderly in day to day routine tasks or providing orientation in a building for new visitors. Thus, the dialog manager was programmed to configure itself from micro-world specification files containing definitions of the concepts one can speak about and definitions of the first-order predicates that are true in the micro-world.

Keywords: Dialog manager, Micro-worlds, Pepper, Prolog, First-order logic, Fuzzy matching.

1. Introduction

The expectation that people can simply talk to computers and receive spoken answers in their native language is nowadays commonplace among Artificial Intelligence (AI) public who has been exposed to science fiction movies and social media advertisements to a large extent. This impression is greatly amplified by the "intelligent" assistants which can be found on any smartphone and to which one can pose questions and get replies in spoken natural language. What most people do not realize right away is that the more complex the question gets, the more often the "default" answer is offered, such as a standard web search for the smartphone assistant or a type of a "Please rephrase the question" answer for a task-oriented dialog system. Furthermore, a coherent dialog with multiple discussion threads, tackling Natural Language Processing (NLP) problems such as ellipsis or coreference resolution is out of the scope of most commercially available dialog systems. It happens because their goal is not to advance the knowledge in the Natural Language Understanding (NLU) field of AI, but to fulfil the immediate (i.e. frequent as determined by the company owning the dialog system) needs which include setting reminders, reserving flights, retrieve information on specific subjects such as movies or celebrities and so on.

That is to say that, currently, there is no general purpose computer dialog system able to sustain a dialog on any given subject as a human can (though there are computer programs called "chatbots", striving to achieve this goal, none of them is yet able to fool someone into thinking that she/he is speaking to another human being). But there are specialized, state-of-the-art computer dialog systems able to communicate on a given topic very well, close to human performance (termed "task-oriented dialog systems"; see Chapter 26 of Jurafsky & Martin, 2019). For example, there are tutoring systems that can substitute a teacher explaining a student how to solve a physics problem (Rus et al., 2013) or dialog systems that can assist a doctor with clinical assessments (Campillos-Llanos et al., 2020).

The task-oriented dialog system presented in this paper is called ROBIN Dialog Manager (RDM) and it was (and actively is) developed in the ROBIN Dialog project (Tufiș et al., 2019; Anon, 2018). This project concerns itself with Romanian language technologies deployment on the Pepper robot (SoftBank Mobile Corp., 2014; SoftBank Robotics Europe – SAS (Limited Company), 2020), enabling it to perform certain tasks using interaction into spoken Romanian. A task is modeled by defining its universe of discourse in a *micro-world* which is basically a collection of logic definitions coupled with programmed robot behavior necessary to perform the task at hand. For instance, to provide orientation instructions to new visitors in a building, Pepper should know the location of different rooms on its internal map (e.g. programmed robot behavior that enables the robot to navigate to a specific room) together with a logic associated with rooms (e.g. what happens in each room). Thus, equipped with this micro-world specification, Pepper can answer questions

https://doi.org/10.24846/v29i4y202003

such as "Where is the Python programming course taking place?" with e.g. "Room 209. Let me take you there."

In what follows, Section 2 surveys the existing literature on task-oriented dialog systems using natural language to First-Order Logic (FOL) translation (the reasons why this strategy was adopted for RDM will be explained below). Section 3 presents possible dialog scenarios that were taken into consideration for RDM development while Section 4 briefly introduces the Automatic Speech Recognition (ASR) and the Text-To-Speech (TTS) modules for Romanian that enable Pepper to interact with users in spoken Romanian. Finally, Section 5 details the micro-world composition and the inner loop of RDM. It includes the question analysis and logic predicate extraction and argument unification, augmenting the presentation with specific examples. Additionally, Section 5 explains how it maps the user's request to a robot behavior which can be either verbalizing the response or executing a task related to the robot planning (e.g. going from one location to another). Section 6 provides the conclusion of this work and offers suggestions for a future research.

2. Related Work

One very recent research direction concerning Question Answering (QA) systems, amenable to be adapted to task-oriented dialog systems is the use of the pre-trained BERT language models (Devlin et al., 2019). BERT is a general-purpose sentence to a real vector deep neural network encoder (Vaswani et al., 2017), running with hundreds of millions of parameters. It is first pre-trained to predict words in context on very large corpora and then, among other tasks, fine-tuned to predict the correct answer to a question in a given input text, on a quite big set of question/answer pairs. For example, for the question "*Who* was **Albert Einstein**?" and the text snippet "**Albert Einstein** was *a German-born theoretical physicist* who developed the theory of relativity.", the QA fine-tuned BERT is able to tag "a German-born theoretical physicist" as the correct answer for the question. BERT can, in principle, be used to implement a task-oriented, one-turn (only one question followed by the final answer), dialog system, provided that 1) the set of possible question/answers pairs for the task exists (and is large) and 2) the details of the task remain fixed and cannot be updated during the execution of the task.

The BERT-like approaches were considered for the design of RDM, but the lack of the question/answer pairs dataset and the (soft) requirement that the universe of discourse could be updated on the fly, while the robot executes the task, pointed to a more traditional, yet more flexible approach: question analysis and its automatic translation to a predicate-argument structure followed by fuzzy predicate matching with *true* predicates in the universe of discourse. Thus, RDM draws from two frameworks for task-oriented dialog systems: Prolog-style theorem proving and, mainly, from slot-filling dialog systems.

(Smith et al., 1995) is a prototypical example on how *Prolog-style theorem proving* can work in a task-oriented dialog system: guiding the user to perform a complex task by exchanging clarification questions aimed at establishing the true facts about the state of affairs. Question analysis is done using a rigid grammar, specifically designed to map natural language assertions to FOL predicates. A task is defined in Prolog as a conjunction of subgoals that must be satisfied so that the task is successfully fulfilled. The dialog controller remembers which goal is currently being proven and an "Interruptible Prolog Simulator" keeps track of partially proven goals. Thus, goal fulfillment can be done as the dialog unfolds, not necessarily in a sequential manner.

The framework that RDM works with is the *slot-filling* framework for task-oriented dialog systems (Henderson, 2015). Henderson defines a *slot* as being an attribute of an entity that can be stored in a database. Slots can be *informable* (can be used to constrain the search in the database) and *requestable* (the user wants to know the value of such a slot). Slots can take values in finite sets and the *dialog state* at a given turn consists of 1) *goal constraint* – the set of assigned values of the informable slots, 2) *requested slots* – the list of slots which user wants to be informed about and 3) *dialog search method* – the way the user is interacting with the dialog system: by constraints (the user is trying to narrow down the search), by alternatives (the user is requesting alternative information if what she/he wants is not available) or finished (end of conversation).

3. Dialog Scenarios

Characteristics

Programming a robot like Pepper to perform a task implies a first step of choosing the "scenery" for the task, defining the task and "teaching" the robot the necessary communication skills, when verbal interaction with people is required for the task to be performed. This mimics a screenplay: the actions are identified, the participants to the actions are specified, the circumstances of each activity are defined: place, time, relations between actions, etc. All actions of the robot are the result of a correct understanding of the human-robot communication. This screenplay is called a *micro-world*: that is, the task formal definition (in the foreseen circumstances) together with the actions the robot can take, which are discrete and already inventoried in the planning component of the robot.

In the present use case, it is assumed that only one person can speak with Pepper at a time and Pepper will complete the task for the current interlocutor before greeting and helping another user. The identity of the interlocutor is irrelevant but the ASR system has to have the lowest word error rate (WER) possible, irrespective of the speaker, and has to be able to work in noisy conditions (although Pepper is equipped with directional microphones that, if spoken to properly, will eliminate most of the background noise).

Example screenplay 1: orientation assistant in a university building

One of the screenplays casts Pepper as host in one of the university buildings. The micro-world contains information about the rooms in the building, their location within it, and it is expected to navigate from its current location in the building to a certain room. It also contains the timetable: titles of the courses taking place in each room, teachers of the courses, their dates, day and time, their structure (lectures, seminars, laboratories, exams, etc.) thus enabling it to answer questions about all of these. Besides courses, the administrative offices are also relevant for students. That is why, information about them (location, working hours, staff, etc.), as well as (about) the documents (and their electronic format or location) involved in the students' interaction with these offices must also be accessible to the robot so as to answer questions and even make documents of interest available to interested parties. An example of a dialog in this micro-world could be the following, though in Romanian:

User: Hi, Pepper!

Pepper: Hello! How can I be of assistance?

User: Where is the Python programming course taking place?

Pepper (*supposing that there are two such courses taking place at the same time, in different rooms*): Who is your teacher?

User: Cristina Iacob.

Pepper (*knowing where Cristina Iacob teaches her course*): Room 209. Do you want me to show you where that is?

User: Yes! Thank you.

Example screenplay 2: sales assistant for laptop acquisition

In another micro-world Pepper is a sales agent in the laptops department of an electronics store. The screenplay imagined in this micro-world focuses on the dialog between the robot and a potential buyer, with Pepper verbalizing answers.

The dialog between Pepper and the customer is centred around three main cases. In the first case the customer already knows what she/he wants. In this situation the potential buyer will provide to Pepper the laptop specific model or the key characteristics that will identify the desired laptop and Pepper can inform the buyer if the laptop is in stock or not.

In the second case, the customer only knows partial details of the characteristics of the desired laptop. She/he will tell Pepper some details of the laptop. For instance, the customer can give a precise configuration for the laptop display, weight and battery autonomy from which Pepper will build a short list of several options. For other categories or specifications, the customer will give a description of what she/he needs such as the newest processor on the market available or the biggest RAM capacity. This will enable Pepper to filter down the list of laptops and come up with a recommendation.

In the third case, the customer does not really have a specific configuration in mind and can only give

an overall indication of her/his need. For instance, the customer might want a laptop for gaming or home use. What is specific in this situation is that Pepper will build a list of compatible laptops and will proceed with a laptop recommendation.

In all situations Pepper can help customers locate the items that they are looking for, call for help and answer FAQ, like the situations where the stock availability or the actual location are relevant for the buyer. An example dialog in this micro-world could be the following:

User: Hi, Pepper!

Pepper: Hello! How can I help you?

User: I want to buy a Dell laptop with Windows 10.

Pepper: What is the RAM specification?

User: 8 GB.

Pepper: Do you need a dedicated video card?

User: No, I don't think so.

Pepper: I recommend the Dell Latitude 5480 that is in stock.

User: Thanks! I will check it out.

4. Speech Interfaces

To hear and speak Romanian, Pepper needs access to Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) modules. These modules will be briefly presented below. Currently, RDM uses the ASR and TTS modules as web services but they could be integrated into the Java codebase for better responsiveness.

ASR module

The ASR module used by RDM employs a classic Hidden Markov Model with Gaussian Mixture Models ASR algorithm that builds acoustic models and language models to Viterbi-decode the best transcription from speech. The module uses the Kaldi ASR toolkit (Povey et al., 2011).

The 3- and 4-grams language models (LMs) were trained on a 592 million words sub-corpus of CoRoLa: the 3-gram model was used for decoding and the 4-gram model was used for rescoring candidates. The language models were built with the SRILM language modeling toolkit (Stolcke et al., 2011), employing Chen and Goodman's modified Kneser-Ney discounting and interpolation for all n-grams, offered as options by SRILM.

The acoustic model uses a phonetic lexicon of more than 2 million Romanian word forms extracted from the unigrams of the 4-gram LM. These word forms were phonetically transcribed using the algorithm described in (Stan et al., 2011). There are 26 Romanian phonemes together with 4 "silence" phonemes. The training corpus has 73.4 hours of speech and the test corpus has 24.6 hours of speech and both corpora come from our Romanian reference corpus CoRoLa (Barbu Mititelu et al., 2019). The word error rate (WER) of the ASR module is 30.2% on the test corpus, but about 10% when measured empirically on transcribing questions from the micro-worlds exemplified in Section 3.

The ASR module used by Pepper can work in two scenarios: online and offline. The one that is currently used in RDM is the offline scenario, which means that the entire audio file that needs to be transcribed is provided in one big chunk to the ASR system. This approach is best suited for short, concise phrases. The second one, the online decoding, would require more hardware resources, both in edge and cloud environments. This one is more fitted for live transcriptions and translation due to the streaming manner of generating and interpreting audio data by the ASR. The online decoding requires some spoken context until the first output block is generated, which can lead to a bigger first-time response of the system.

One much better alternative that is currently under experimentation is the end-to-end Romanian ASR module based on the Deep Speech 2 neural network (Avram et al., 2020). This ASR module, which is trained on a 230 hours Romanian speech corpus, manages to obtain a 9.9% WER on its test set, while it transcribes, on average, in 70 milliseconds per utterance. For comparison, the Kaldi ASR response time for average sized utterances (less than 15 spoken seconds) follows a 1:2 ratio (10 spoken seconds will delay the transcription by 20 seconds). The replacement of the Kaldi ASR module with the Deep Speech 2-based one is subject for future work.

Besides the low WER and response time, one of the major challenges in usable ASR is the signal-to-noise ratio of the utterance recording, given the

fact that the module is expected to work in noisy environments. This paper aims to solve this problem by using sensitive, unidirectional microphones installed on Pepper, coupled with training the Deep Speech 2 network on noisy recordings.

Another challenge for the ASR module is to correctly identify the brand names associated with different properties of computing hardware (e.g. a "Corsair" memory stick). Thus, to adapt the ASR module to the "sales assistant" scenario mentioned in Section 3, more than 200 questions were created, containing laptop properties (e.g. memory, storage, etc.) with their associated brand names (e.g. Corsair, SanDisk, etc.) The full set of questions has been audio recorded in six voices (3 males and 3 females) and then added to the ASR training corpus. All the considered brand names have Romanian phonetic transcriptions added to the phonetic lexicon (e.g. IBM – [aibiem], Dell – [del], etc.)

TTS module

The TTS module implements a two-step synthesis, similar with modern approaches such as Tacotron (Wang et al., 2017), Char2Wav (Sotelo et al., 2017) and ClariNet (Ping et al., 2019). The two-step algorithm can be seen as an autoencoder, with the first step transforming the input (the text) into a hidden representation (Mel-frequency cepstral coefficients (Logan, 2000)), while the second step transforms the hidden representation back into the input (or audio waveform in the case of TTS).

To achieve *the first step*, one-hot vector encodings for the characters in the text (both letters and special characters such as space, comma, and period) are created. The one-hot encoding of the current character goes through a Bidirectional Long Short-Term Memory (BiLSTM) layer and its output is concatenated with the speaker encoding, thus obtaining an *intermediate vector representation*. Lastly, a pipe of an LSTM layer and three Fully Connected (FC) layers (called the Sequence-To-Sequence (Seq2Seq) model) are used to predict the Mel-frequency cepstral coefficients from the intermediate vector representation.

The Seq2Seq model is enhanced by an Attention Block (AB) whose role is to focus the input sequence around the most important elements for the current prediction. For example, one does not need all the elements in the input to predict the beginning of the waveform, but only its first elements. More precisely, the AB is a neural network that receives the LSTM state and computes its probability of relevance, at time step t, for the current prediction. The attention-enhanced, current LSTM state is obtained by scalar-multiplying the LSTM states at time steps 1 to t with their corresponding probabilities given by AB and then summing all of them.

While AB was trained as previously described, at run-time the AB network is used in a different manner: always use *consecutive, most relevant* (as given by the AB block) LSTM states for inference (do not scalar-multiply and sum all LSTM states). If the previous, most relevant LSTM state was at time step s and the current, most relevant LSTM state is at time step $s + 1$, it is used, otherwise the LSTM state at time step s is still used for the current prediction. Using the consecutive condition greatly increases the quality of audio, but also the synthesis duration.

Finally, the previous attention-enhanced LSTM state and the previous prediction are used as input for the LSTM layer at present time step. The current attention-enhanced LSTM state is fed to a pipe of three FC layers to get the current prediction. During training, the neural network predicts until there are as many predictions as in the ground truth. However, during inference prediction is performed until the AB says, five times in a row, that the same LSTM state is the most relevant. The number five was chosen empirically, as AB says, on average, three times in a row that the same LSTM state is the most relevant.

For *the second step* the ClariNet implementation is used to obtain the audio waveform from the Mel-frequency cepstral coefficients.

The main advantage of this TTS module is that it is an end-to-end system that can synthesize words it was not trained with. Also, the fact that it uses character encodings instead of phone encodings provides an advantage since it does not need to compute any phonetic transcription of the input text. On the other hand, lacking the extra information provided by the phonetic transcription, some words may be wrongly synthesized. One such example is the mute/whispered vowel "i" in words like "București" or "Ploiești" which can be fully heard in the synthesized speech, which effectively moved the accent to the last syllable.

The TTS module uses deep neural networks with a lot of parameters and, as such, it needs to be run on a GPU to produce results in real time. If so run, its synthesis time can take as much time as the time needed to utter the input text, again a 1:1 ratio as with the ASR module.

5. ROBIN Dialog Manager

RDM (Ion, 2020a) is a Java-based dialog manager that automatically constructs its universe of discourse from a micro-world specification file. As stated before, a *micro-world* is a collection of definitions about the things one can speak of in that micro-world coupled with associated named robot behavior to be used in subsequent robot planning. The dialog manager is expected to infer and provide the *contextually parametrized user intention* to be used by the robot planning programming team to complete the action that is requested by the user. This action can be, for instance, simply a verbalization of the answer, tapping into the TTS module of Pepper or a more involved action that requires the robot to navigate somewhere, lift and offer objects, recognizing persons using facial recognition, delivering messages to them and so on.

Micro-world file specification

A micro-world file (for example, look at the `precis.mw` file (Ion, 2020b) from the GitHub repository) contains the definitions of concepts and predicates one can speak of.

A *concept* is a common noun that is a *typed set* of known objects in the universe of discourse. A concept is defined as e.g.:

```
CONCEPT sală, laborator, cameră -> LOCATION
```

where the first word **sală** (English "hall") is the canonical name (in lemma form) of the concept, followed by its possible synonyms (English "laboratory" and "room"). This "synset" (not exactly a synset in the WordNet (Fellbaum, 1998) sense, but rather a semantic neighborhood, including synonyms, hyper- and hyponyms) is of type LOCATION, meaning that this type can be matched in questions such as "**Unde** se desfășoară…" (English "**Where** does the…") or "În **ce cameră** se desfășoară…" (English "In **what room** does the…").

The enumeration of the typed set introduced by the concept definition is done with the *reference* definition, such as:

```
REFERENCE sală sala 209 = S1
REFERENCE sală sala de consiliu = S3
REFERENCE curs laboratorul de informatică = C1
```

Thus, for a concept identified by its canonical name (e.g. **sală**), instances are listed (e.g. "sala 209") together with aliases to be used in predicate definition (e.g. S1). These are specific phrases that the RDM expects to encounter/can offer as answers in a conversation in this micro-world.

There are three predefined types: LOCATION, PERSON and TIME together with a default type, WORD, which is, in fact, the concept canonical name as in e.g.:

```
CONCEPT curs, materie, seminar, laborator -> WORD
```

Thus, in a question such as "Unde se desfășoară **cursul** de sisteme de operare?" (English "Where is the OS **course** taking place?") the noun phrase "**cursul** de sisteme de operare" is of type **curs**, as its head (in lemma form) is listed as a concept of type WORD.

References of types PERSON and TIME can be added separately, with definitions such as:

```
TIME marți la 8:00 = T1

PERSON Magda Vlad = P2
```

where the instance of the type PERSON (e.g. Magda Vlad) is followed by the reference alias (e.g. P2) to be used in the predicate definition. Similarly, the time instance "marți la 8:00" (English "Tuesday at 8:00") is referenced by means of the alias T1.

The *predicate* is the unit of information that Pepper knows it is true. A predicate can hold true on a variable number of typed references, defined as previously described. Checking for types and number of arguments (order is not important) helps RDM to resolve ambiguities among predicates with the same name. Predicates are defined using alias names of previously defined references, e.g.:

```
TRUE ține C1 S1 T1 P2
```

This means that the course C1 ("laboratorul de informatică") of type **curs** takes place (predicate canonical name **ține**) in room S1 ("sala 209") of type LOCATION at time T1 ("marți la 8:00") of type TIME and it is lectured by professor P2 ("Magda Vlad") of type PERSON. Furthermore, to allow synonymic variation of the predicate name and *to map predicates to users' intentions*, one may write e.g.:

```
PREDICATE ține, desfășura, preda ->
SAY_SOMETHING
```

Here, **ține** is the canonical name of the predicate followed by a synonymic series of possible formulations, much as in the case of concept definitions. This predicate signals a user intent of learning some information, causing Pepper to verbalize its response ("SAY_SOMETHING"). This is the mechanism that maps predicate realization to actions that are accomplished by the planning component of the robot (SAY_SOMETHING is the only action that is fulfilled by RDM, calling its TTS module and speaking the answer).

Question analysis

Question analysis is realized through 1) **pre-processing the input question** by tokenizing, part-of-speech tagging, lemmatizing, and dependency parsing it and 2) **analysing the predicate-argument structure** induced by the parse tree.

Text pre-processing is offered by the TEPROLIN web service (Ion, 2018) which is integrated into the RELATE platform (Păiș et al., 2020a; Păiș, 2020b), a portal for the dissemination of Romanian language technologies. Following the pre-processing step, *assuming there is only one main verb in the question*, the question analysis step will attempt to construct a predicate from the user's question by performing the following operations:

1. If there is no main verb in the question (either the user did not ask a complete question or the POS tagging failed to identify a main verb) and if there is no context (the dialog is just beginning) return a "Please rephrase the question" response. If the dialog manager stores a partially bound predicate from the previous question and the current question does not have a main verb, use the previously bound predicate to reply (see the example presented in the "Inference module" section below).

2. Identify the root of the question from the parse tree as the main verb. Take its lemma to be the canonical name of the predicate.

3. Take the list of noun, prepositional or adverbial phrases that are directly linked to the main verb as arguments of the predicate. Predicate arguments can only be introduced by dependency relations that designate subject, object, or complement dependents.

4. The argument that contains an interrogative adverb, determiner, or pronoun (in Romanian this is always the first phrase in the question) is the unbound (to be resolved) typed variable to be unified by the predicate matching algorithm. Its bound value is offered back as the answer to the question.

Take for instance the question "În ce sală se ține laboratorul de informatică?" (English "In what room is the informatics laboratory taking place?") Given this question, the text pre-processor will parse it into the tree from Figure 1.

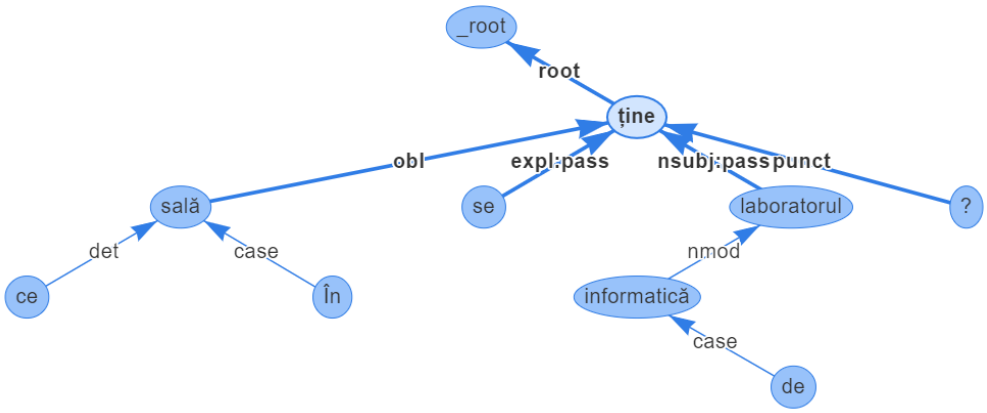

Figure 1. The parse tree for the question "În ce sală se ține laboratorul de informatică?"

The main verb of the question is **ține** which is the predicate name. Its arguments are the following:

1. The oblique prepositional phrase "În ce **sală**" which is the requested variable *X* of type LOCATION (see the concept **sală** defined above) because it is modified by the interrogative determiner "ce".

2. The subject noun phrase "laboratorul de informatică" which is a reference of type **curs** (see reference definition with alias C1 above).

With this information, the question analysis module constructs the partially bound predicate

```
ține "laboratorul de informatică"/
     curs X/LOCATION
```

which it passes to the inference module whose job is to resolve the reference of *X*.

Inference module

Presented with a partially bound predicate, it is the job of the inference module to find the appropriate answer to the posed question or ask supplementary, clarifying questions. This is accomplished within the main RDM question-answering loop.

Predicate and argument fuzzy matching, that is, *the identification of the most likely predicate* from the micro-word definition that was referred to in the user's question, is done with the following algorithm:

1. The predicate name is searched for, either verbatim or through a synonym, in PREDICATE definitions of the micro-world definitions and all predicates (TRUE definitions) that match the predicate name received from the question analysis module are retrieved. If no such predicate is found, return a "Please rephrase the question" response.

2. From the set of true predicates, select the one such that:

    a. The sum of its arguments fuzzy-match scores is maximum, while *strictly enforcing the type equality* of the fuzzy-matched arguments.

    b. The unbound variable has the same type with an existing argument.

3. If Step 2 produced a predicate, *answer with the bound value* of the best matched predicate.

Given two lists of POS-tagged and lemmatized words *D* (description) and *R* (reference), the fuzzy-matching algorithm, mentioned at Step 2.a. above, works in the following way, for all word pairs d_i from *D* and r_j from *R* such that both d_i and r_j are content words (i.e. nouns, verbs, adjectives or adverbs) and *i* and *j* are the positions of words in their respective lists:

1. If lemma of d_i string-equals lemma of r_j or if lemma of d_i loosely-equals lemma of r_j, given the semantic neighborhoods (synonyms and direct hyper- and hyponyms as defined by the Romanian WordNet (Tufiș et al., 2013), accessed from the RELATE platform) of d_i and r_j , add $|i - j| + 1$ to the matching score $S_{d\to r}$ and advance to the next *i* index.

2. If no matching can be found at Step 2 and if *lss* is the Levenshtein similarity score between the lower-cased d_i and r_j , add $lss \cdot (|i - j| + 1)$ to the matching score $S_{d\to r}$ and advance the next *i* index.

It follows that $S_{d\to r}$ (from description to the reference) cannot be larger than the size of the *D* list and the score $S_{r\to d}$ (in the reverse direction) cannot be larger than the size of the *R* list. Thus, the symmetrical matching score

$$S = 2 / (S_{d\to r} + S_{r\to d})$$

is at most 1 when *D* and *R* are identical, with respect to their content words.

To exemplify the matching algorithm for the question "În ce sală se ține laboratorul de informatică?", as already presented, the question analysis module retrieves the partially bound predicate

```
ține "laboratorul de informatică"/
     curs X/LOCATION
```

Considering that the micro-world contains the true predicate definition:

```
TRUE ține C1 S1 T1 P2
```

in which C1 is "laboratorul de informatică" of type **curs** (matching score $S = 1$) and S1 is "sala 209" of type LOCATION, and strictly observing type matching, *X* can be unified with S1 and the response "Sala 209." is available to the user.

The dialog manager loop always keeps track of the last, best-matching predicate and its argument bindings. If the user wants more information with respect to this predicate, subsequent predicate matching happens in this context. Resuming the given example, if the user's next question would be "Cine îl predă?", the question analysis module provides the analysis

```
preda Y/? X/PERSON
```

in which any personal pronoun is transformed into an unknown type variable and the special interrogative pronoun "cine" (English "who") is only applicable to persons. Now, because **preda** is a synonym of **ține**, the last best-matching predicate with bound arguments C1 and S1 is able to offer the answer because P2 has the same type as our requested variable *X* (PERSON): "Magda Vlad." *Y* cannot be reliably bound as more detailed syntactic analysis would be needed. This is a simplified way of solving elliptic questions because it still requires the presence of a main verb. A further enhancing of this inferential process (but fuzzier) is to assume, *in case no main verb is present in the question*, that the previous best-matching predicate would apply. This way, the follow-up question "La ce oră?" (English "At what time?") could be interpreted as:

```
ține X/TIME
```

Keeping track of the last best matching predicate and its argument bindings, the elliptic question will be correctly answered with "marți la 8:00", as the best-matching predicate **ține** is able to bind its last argument to T1.

6. Conclusion

The ROBIN Dialog Manager is a Java-based, configurable dialog manager ready to be integrated into any application that requires interaction into spoken Romanian. One of its advantages is that, with its smartly abstracted class hierarchy, it can be extended *to handle other languages*, such as English. Thus, if the language pre-processing chain doing tokenization, part-of-speech tagging, lemmatization and dependency parsing is available along with ASR and TTS modules, RDM can work in any language, provided that the required Java implementations using the targeted language technologies are supplied.

Another advantage of RDM is *the micro-world definition file* with which it can handle a new dialog scenario, requiring the knowledge engineer to define concepts, concept references and predicates that are applicable to the new micro-world. The robot planning algorithm can utilize the resolved true predicate instance with its bound arguments in ways that are suitable for the actions to be performed following the conversation with the user. Furthermore, named robot behavior can be attached to any predicate definition in the micro-world file to map speech to action taking.

One disadvantage of the proposed solution is that the knowledge engineer is tasked with the complete definition of the micro-world with the caveat that any incomplete specification (e.g. the right synonyms are not provided or not all the possible references are enumerated) will cause RDM to not work. While the solution to this problem is straightforward (just add the missing information), it is conceivable that *a complete specification might never be attained*.

RDM will receive some further improvements such as the ability to populate its universe of discourse from external sources. If the relevant information is stored in e.g. a database, the ability of creating predicate instances directly from the database, via an Internet connection, will be added. Thus, the knowledge base of RDM could be updated on the fly, without disrupting the functionality of the dialog manager.

Acknowledgements

The research described in this article 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

Anon, (2018). *Website of the ROBIN Project*. Available at: <http://aimas.cs.pub.ro/robin/robin-dialog/>, last accessed: October 25th, 2020.

Avram, A. M., Păiș, V. & Tufiș, D. (2020). Towards a Romanian end-to-end automatic speech recognition based on Deep Speech 2. In *Proceedings of the Romanian Academy*, *Series A, Volume 21*(4). ISBN 1454-9069, in print.

Barbu Mititelu, V., Tufiș, D., Irimia, E., Păiș, V., Ion, R., Diewald, N., Mitrofan, M. & Mihaela, O. (2019). Little Strokes Fell Great Oaks. Creating CoRoLa, The Reference Corpus of Contemporary Romanian, *Revue roumaine de linguistique*, *LXIV*(3), 227–240.

Campillos-Llanos, L., Thomas, C., Bilinski, É, Zweigenbaum, P. & Rosset, S. (2020). Designing a virtual patient dialogue system based on terminology-rich resources: Challenges and evaluation, *Natural Language Engineering*, *26*(2), 183–220.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *arXiv:1810.04805 [cs.CL]*.

Fellbaum, Ch. (1998, ed.). *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Henderson, M. (2015). Machine learning for dialog state tracking: A review, In *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing*, September 19–20, Aizu, Fukushima, Japan. Available at: <https://research.google/pubs/pub44018/>.

Ion, R. (2018). TEPROLIN: An Extensible, Online Text Preprocessing Platform for Romanian. In *Proceedings of the International Conference on Linguistic Resources and Tools for Processing Romanian Language (ConsILR 2018)*, November 22-23, 2018, Iași, Romania (pp. 69-76).

Ion, R. (2020a). *ROBIN Dialog Manager, Open-source software*. Available at: <https://github.com/racai-ai/ROBINDialog>.

Ion, R. (2020b). *An example micro-world file*. Available at: <https://github.com/racai-ai/ROBINDialog/blob/master/src/main/resources/precis.mw>.

Jurafsky, D. & Martin, J. H. (2019). *Speech and Language Processing*, 3rd ed. draft. Available at: <https://web.stanford.edu/~jurafsky/slp3/>.

Logan, B. (2000). Mel Frequency Cepstral Coefficients for Music Modeling. *In Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, October 2000, Plymouth, USA (pp. 1-11).

Păiș, V., Tufiș, D. & Ion, R. (2020a). A Processing Platform Relating Data and Tools for Romanian Language. In *Proceedings of the 12th Language Resources and Evaluation Conference*, European Language Resources Association, Marseille, France (pp. 81–88).

Păiș, V. (2020b). *The RELATE website*. Available at: <https://relate.racai.ro>.

Ping, W., Peng, K. & Chen, J. (2019). ClariNet: Parallel wave generation in end-to-end text-to-speech, *arXiv preprint arXiv:1807.07281.*

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G. & Vesely, K. (2011). The Kaldi Speech Recognition Toolkit. In *Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, Hilton Waikoloa Village, Big Island, Hawaii, US, IEEE Catalog No.: CFP11RW-USB. Available at: <http://www.danielpovey.com/files/2011_asru_kaldi.pdf>.

Rus, V., D'Mello, S., Hu, X. & Graesser, A. (2013). Recent Advances in Conversational Intelligent Tutoring Systems, *AI Magazine*, *34*(3), 42–54.

Smith, R. W., Biermann, A. W. & D. Richard Hipp, D. R. (1995). An Architecture for Voice Dialog Systems based on Prolog-Style Theorem Proving, *Computational Linguistics*, *21*(3), 281–320.

SoftBank Mobile Corp (2014). *SoftBank Mobile and Aldebaran Unveil "Pepper" – the World's First Personal Robot That Reads Emotions*. Available at: <https://www.softbank.jp/en/corp/group/sbm/news/press/2014/20140605_01/>, last accessed: October 25th, 2020.

SoftBank Robotics Europe - SAS (Limited Company) (2020). *Pepper robot presentation website.* Available at: <https://www.softbankrobotics.com/emea/en/pepper>, last accessed: October 25th, 2020.

Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A. C. & Bengio, Y. (2017). Char2Wav: End-to-End Speech Synthesis. In *Proceedings of the International Conference on Learning Representations – ICLR 2017* (pp. 1-6).

Stan, A., Yamagishi, J., King, S. & Aylett, M. (2011). The Romanian Speech Synthesis (RSS) corpus: building a high quality HMM-based speech synthesis system using a high sampling rate, *Speech Communication, 53*, 442–450. DOI: 10.1016/j.specom.2010.12.002

Stolcke, A., Zheng, J., Wang, W. & Abrash, V. (2011). SRILM at Sixteen: Update and Outlook. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, *Dec. 2011, Vol. 5.*

Tufiș, D., Barbu Mititelu, V., Irimia, E., Mitrofan, M. Ion, R. & Cioroiu, G. (2019). Making Pepper Understand and Respond in Romanian. In *Proceedings of the 22nd International Conference on Control Systems and Computer Science* (pp. 682 – 688). DOI: 10.1109/ CSCS47589.2019

Tufiș, D., Barbu Mititelu, V., Ștefănescu, D. & Ion, R. (2013). The Romanian Wordnet in a Nutshell, *Language Resources and Evaluation, 47*(4), 1305–1314.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems - NIPS 2017*, Long Beach, CA, USA (pp. 6000–6010).

Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R. & Saurous, R. A. (2017). Tacotron: Towards end-to-end speech synthesis, *arXiv preprint arXiv:1703.10135*.