July 2020

COMBINED APPROACH FOR BUILDING A DOMAIN-SPECIFIC NATURAL LANGUAGE SEMANTIC PARSER

Michael Laor

Tal Maoz

Aviva Vaknin

COMBINED APPROACH FOR BUILDING A DOMAIN-SPECIFIC NATURAL LANGUAGE SEMANTIC PARSER

AUTHORS:
Michael Laor
Tal Maoz
Aviva Vaknin

ABSTRACT

Proposed herein is a combined approach for building a domain-specific natural language semantic parser. The disclosed technique uniquely combines several known approaches with a new, original rule-based approach to obtain accurate mappings of utterances in a given domain into a single, pre-defined, domain-specific data structure that can be used for further processing. An implementation of the semantic parser has been built for the networking domain, including the QoS, filter (ACL) and set-path sub-domains, and has been integrated with a dashboard-managed L3 firewall for end-to-end implementation of filtering commands.

DETAILED DESCRIPTION

In general, for a pre-known, given domain, an objective is to map intentions, directives and queries spoken using natural language (NLP) into a specific data structure defined over that domain. Specifically, a working, concrete method/algorithm is presented to map a natural language utterance into a pre-defined data structure that may be used in a generic manner for further processing. Among other domains, this is suited for the networking space, where potential use cases include configuration, queries, telemetry, and others. A higher abstraction of the interaction with the network is enabled, which can be advantageous in several ways:

- it allows a higher level of abstraction and hence more efficiency to operational personnel when configuring, maintaining, and debugging the network; and
- it allows non IT personnel (e.g., small- and medium-sized business (SMB) customers, factory operational personnel, etc.) to interact with the network using natural language, telling the network its desired behavior, hence saving the IT personnel cost and increasing their operational efficiency.

Other proposals for a natural language front end for a system involve defining an intuitive user interface for one particular system, mapping out the translation within the user interface and using action templates. In contrast, while the idea to create a natural language user interface for a technical system is similar, the proposal described herein approaches the problem in a completely different way. Specifically, the described technique requires no pre-mappings and no hard-coded templates. Rather, the proposed system employs a general-purpose semantic parser based on an original algorithm to build the mappings from natural language input into a single data structure representing the user's intent. That data structure is then used when interfacing with any underlying mechanism to build and carry out the appropriate commands. While a prototype has been built for the networking domain, the system can be trained to build mappings for a variety of systems and domains simply by supplying appropriate data files.

The primary feature of the proposed system is a generic semantic parser that is capable of interpreting a natural language utterance in a pre-defined domain, possibly comprised of several sub-domains, and mapping it into a relevant data structure for further processing. The semantic parser comprises the following six parts:

1. a Deep Neural Network (DNN) based sub-domain classifier;
2. a Natural Language Processing (NLP) engine for part of speech (POS) tagging;
3. Lexical Lists (LLs);
4. a custom trained, DNN based, Named Entity Recognizer (NER);
5. an Adaptive Learning Knowledge Base (ALKB); and
6. a scoring metric.

If the domain comprises sub-domains, the utterance is first classified into its correct sub-domain using the classifier. Next, POS information is obtained using the NLP engine. An LL is a set of keywords lists, patterns expressed as regular expressions, code snippets, and activation rules that together recognize and map single words or phrases to a specific field in the given data structure. It also uses POS information provided by the NLP engine, as well as learned data from the KB. The LL is given in JSON format. It defines its name, domains it applies to and the list of SNR fields it maps to. If a word is found within one of

the keyword lists, it is mapped. A regular expression can define more complicated capture patterns. It can include references to one or more keyword lists to easily build complicated patters. It can also define conditions on capture groups using code snippet references, which can also replace words within the text captured by the regex. Next, a regex defines which capture group contains the actual value(s) that should be mapped into the SNR. Code snippets can be parametrized and allows for more complicated text matching and are also used to access the POS data provided by the NLP engine when needed. Finally, logical (ANR/OR) activation rules can be defined to layout the checkup sequence within the LL. If not given, the default rule is a simple OR over all the lists, regexes and code snippets defined within the LL.

The engine processes an utterance by traversing the sentence word by word. At each word it compares the word with each LL for that domain and attempts to find a rule with the best match. It then sequentially looks at each word-sequence starting with that word, adding one word each time. When a match is found, the results are stored in an SNR. If more than one match is found for the same SNR field, the SNR is duplicated, once with each value, thus generating a set of SNR candidates. Subsequent matches for the remaining fields are updated in each of the SNR candidates. While processing the LL matches, the NLP tree is referred to and the part of speech information is used, attempting to match the longest phrase possible.

In parallel, phrases are mapped from the utterance into the data structure using the DNN NER, employing either a unified domain model or the appropriate sub-domain model if applicable. Since there may be more than one interpretation, all valid mappings are stored, and a scoring metric is used to determine the best combined fit across candidates produced by the LL module and by the DNN on a field-by-field basis. Lastly, the KB is built by asking users to fill in missing information, clarify choices, and confirm predictive mappings by the DNN. The KB stored confirmed predictive mappings with a confidence level such that the more the user confirms the prediction within a given timeframe, the higher the trust. When a certain threshold is passed, the prediction is considered to be world-truth and becomes permanent knowledge.

The proposed system differs from prior solutions in a number of respects. For example, the proposed system is not conversational, as it maps a single complex utterance

into a domain specific data structure rather than parsing the request in stages by using follow-up questions. Furthermore, the semantic parser combines two parsing methods and resolves the two methods for the optimal choice using a novel scoring system. This provides a high level of accuracy, enabling use of the semantic parser operationally. The semantic parser is generic and can be trained for any domain. The approach is resilient to phrasing and dialect variations. Finally, an accumulated knowledge base, collecting user replies, feedback and inputs, is used as part of the semantic parsing process to essentially build world-truth knowledge for a specific domain/sub-domain within a given deployment, thereby forming an adaptive semantic parser.

With the proposed system, users can speak or type their intent naturally, and the intent is deciphered using the network semantic parser. Relevant commands are then issued to the underlying (physical and/or virtual) infrastructure. In the case of network configurations, the user can elect to receive visual feedback that the network has been properly configured.

Recognizing that not everyone is an IT expert, the system enables operations personnel to configure, manage and query the network infrastructure using natural language. This frees up the IT experts to deal with more complex, lower-level infrastructure and issues. Experts too can benefit from the system by enabling them to simply say what they want done, and have the system interpret and carry out the configuration requests, saving costly IT man hours of manually configuring the systems. In one example implementation, the proposed system understands natural language and handles setpath (routing), QoS (quality of service), and filtering (ACL) in the networking domain. A few examples of sentences that can be process by the proposed system are as follows:

Filter domain:

"Please allow all voip to bldg. 10"

"all voip to bldg 10 should be blocked"

"permit udp traffic from 10.0.8.1 to 171.21.58.16/12"

"do not allow traffic from Battle.net"

"all FTP traffic to wireless should be denied"

QoS domain:

"please apply highest priority to Diana's ip-camera packets going to campuses"

"Diana's ip-camera packets outbound to campuses should be sent at highest priority"

"Make all packets of type iso-tp4, marked with de and going towards campuses, oam"

"John's blue ios devices wap secure connectionless session service packets that are flagged with ef and from cellular phone should be sent at lowest latency"

Setpath domain:

"All of Dave's mail going to salesforce.com should go via security broker Lima"

"route all traffic to company.com through the DMZ interface"

"All TCP packets on port 8888 should go through building A26"

Figure 1 depicts a high-level system architecture. A user uses natural language to tell the system what he/she would like done. The request is then translated to an internal request representation, termed *Structured Network Request* (SNR). Next, the system iteratively attempts to resolve addresses and network terms and translate terms relevant to the specific underlying network management system (NMS), asking the user for clarification of ambiguous terms, and continues until the user confirms the entire request. The request is then translated into API calls relevant to the underlying NMS, issues the calls and displays success/failure feedback to the user.

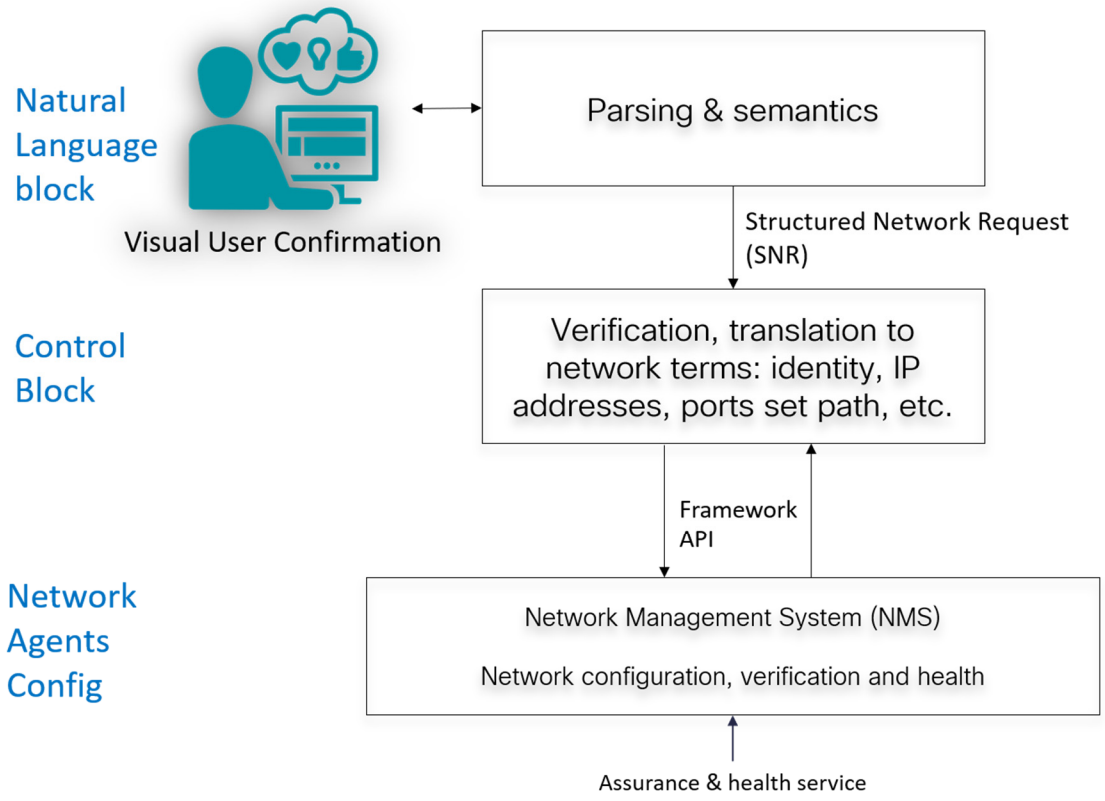

Figure 1: High Level Design

The flow (see Figure 2) begins with the user uttering (or typing) a naturally phrased English (or other language) command to the system.

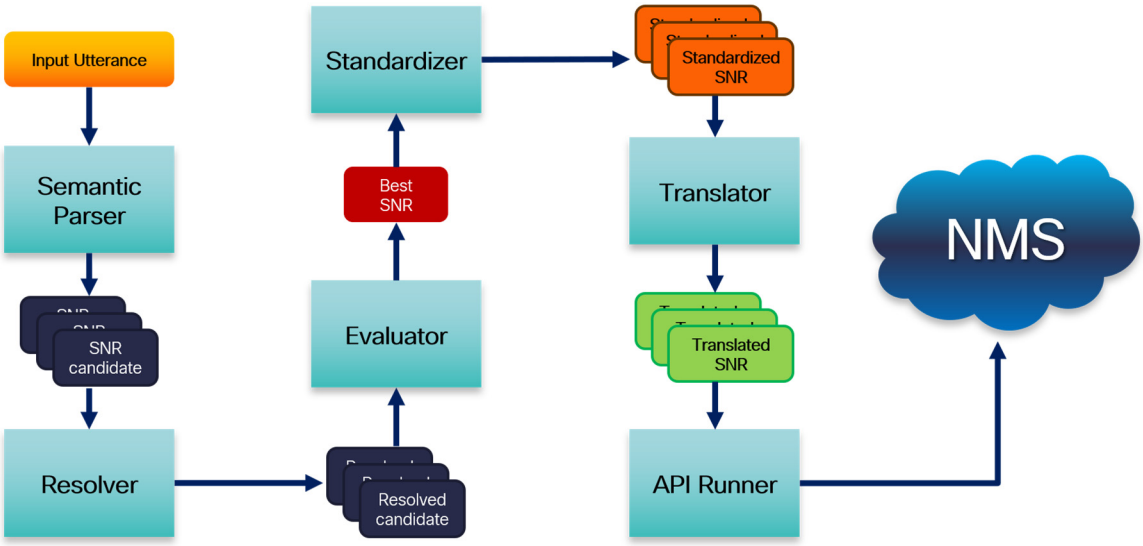

Figure 2: SNR Processing Flow

The semantic parser (see Figure 3), including an NLP engine and a combined algorithmic and Deep Learning mechanism is used to obtain both the grammatical structure and network semantics of the sentence, and to map the network terms into the SNR structure. Since there may be several ways to interpret each sentence, multiple SNRs exist, representing all the possibilities. The set of SNR candidates are sent through a *Resolver* which fills in default values where needed and filters out invalid SNRs according to a pre-defined set of rules. An *Evaluator* then employs a scoring metric to determine the "best" SNR, which is deemed to be the one best matching the user's intent.

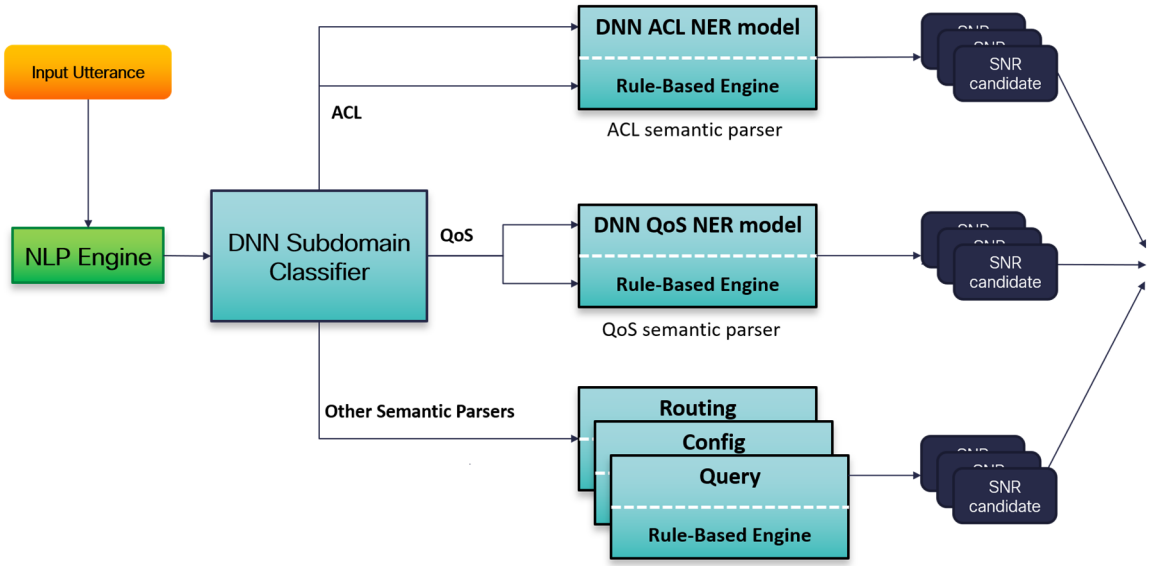

Figure 3: Semantic Parser Architecture

This SNR is then processed by a *Standardizer* which resolves known terms to their equivalent standard forms. This is done using a set of logic rules, which may generate more than one SNR request (e.g. some network service configuration that is issued over TCP and UDP or uses several ports, or a domain begin resolved into a list of IPs), as well as terms learned by the ALKB during previous interactions.

At this point, the now standardized set of SNR requests are translated into the requests that can be interpreted by the selected underlying NMS. A *Translator* performs this job using internal logic and data obtained from the specific network defined by the user.

Both the *Standardizer* and *Translator* may require more information from the user to resolve ambiguous terms, such as unknown identities/locations, vlan/subnet resolution, etc. These questions are presented to the user, and the standardization and translation stages are repeated until all ambiguities are resolved. Where deemed appropriate, the answers are stored in the ALKB for future reference.

Once the user confirms the resolved network request, an *API Runner* builds and issues appropriate API calls to the underlying NMS. Finally, it receives a success/failure notification which is presented to the user.

In internal testing, the above network-space implementation has shown roughly 94% accuracy for the selected sub-domains when compared to gold standard data over 50,000 sample inputs.