EUDAMU at SemEval-2017 Task 11: Action Ranking and Type Matching for End-User Development

Marek Kubis and **Paweł Skórzewski** and **Tomasz Ziętkiewicz**
Faculty of Mathematics and Computer Science
Adam Mickiewicz University in Poznań
{mkubis,pawel.skorzewski,tomasz.zietkiewicz}@amu.edu.pl

Abstract

The paper describes a system for end-user development using natural language. Our approach uses a ranking model to identify the actions to be executed followed by reference and parameter matching models to select parameter values that should be set for the given commands. We discuss the results of evaluation and possible improvements for future work.

1 Introduction

The goal of the end-user development (EUD) is to provide users of software systems with the tools to create, extend or modify software (Lieberman et al., 2006). End-User Development using Natural Language is one of the tasks of SemEval-2017 International Workshop on Semantic Evaluation.

The idea of using tools for software development more accessible than programming languages has been discussed almost since the beginning of computers. The concept of programming with natural language commands and its consequences have been considered already in the 70s (Dijkstra, 1979). However, only in recent years these ideas could be put into practice with the rapid development of advanced natural language processing methods.

A comprehensive overview of recent trends and achievements in EUD has been presented by Paternò (2013). Although most common solution for EUD problem are various variations on graphical user interfaces, solutions using NLP are also present. One of the examples is the Koala (later: CoScripter) system (web browser extension), that uses "sloppy programming", i.e. pseudo-natural language instructions, to automate business processes on the web (Little et al., 2007). Other systems that use simple natural language commands to achieve programming-like goals, described in recent years, include SPOK, an EUD environment for smart homes (Coutaz et al., 2014), and NaturalMash, an EUD tool for creating web mashups (Aghaee and Pautasso, 2014).

This paper describes our system for the end-user development using natural language and reports its performance according to the SemEval 2017 Task 11 evaluation criteria (Sales et al., 2017).

2 Data preparation

We pre-process all the input data: both the action knowledge base and natural language commands. The pre-processing is done in several stages. It includes basic text processing as well as adding lexical features. We use these features to better match the commands to actions. The pre-processor operates on JSON files, in each step adding new fields to the JSON structure so the original data are not lost in the process and can be used in further steps. The input fields we annotate are: `desc`, `name`, `value`, `nl_command_statment`, `provider`, `sample`, `tags` and `api-name`.

The first pre-processing stage is sentence splitting. The sentence splitter is applied to `nl_command_statment` and `desc` fields only.

The second step is tokenization. The input data are amended not only with tokens, but also with token types. The following token types are recognized: text, number, phone number, monetary expression, punctuation, URL, e-mail address, hashtag, file name, other. Types of tokens are recognized using regular expressions. The set of token types has been specified manually in such way that they are useful in the context of both the original training dataset and other possible datasets. The token types are used in further processing stages: anaphora resolution, action detection and parameters matching.

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 1000–1004,
Vancouver, Canada, August 3 - 4, 2017.

In the next step, we append features from SyntaxNet (Andor et al., 2016), Stanford CoreNLP (Manning et al., 2014) and NLTK (Bird et al., 2009) to the natural language commands. In particular, we introduce: part-of-speech tags from SyntaxNet to implement discourse annotation rules (cf. Section 3.1); word lemmata from NLTK for the purpose of preprocessing text for the action ranker (cf. Section 3.2); named entities and constituency parser annotations from CoreNLP (Finkel et al., 2005) for anaphora resolution.

The next pre-processing stage appends features extracted from the user knowledge base. Any reference to the user knowledge base entry that occurs in a command is annotated with the entry identifier and the name of the referred entry field.

The last pre-processing stage is to add anaphora information. Anaphora tags are appended to possible anaphoric terms like *it*, *him* etc. or nouns preceded with definite articles (e.g. *the file*). The antecedent expressions of the anaphora are identified on the basis of their part-of-speech tags, NER tags, token types or phrase categories. For this purpose we use token type annotations and selected features from CoreNLP: tokens, POS tags, NER tags and parse results.

3 System Overview

The main components of the system are presented in Figure 1. A natural language command that has been preprocessed according to the procedure described in Section 2 is passed to the discourse tagger which identifies phrases and relationships among them. Phrases are passed separately through action ranker which identifies candidate actions. Next, reference matcher identifies dependencies that link data values that are returned by the candidate actions with the parameters of their successor actions. Then, the parameter matcher aligns tokens that occur in the phrase to the parameters of the candidate actions that where not linked by the reference matcher. Finally, the phrases annotated with actions and parameter values are passed to the statement mapper which uses the discourse structure to output action instances that conform to the task specification.[1]

[1]For an example of a complete data flow that passes a natural language command through all the system modules, we refer the reader to (Kubis et al., 2017).

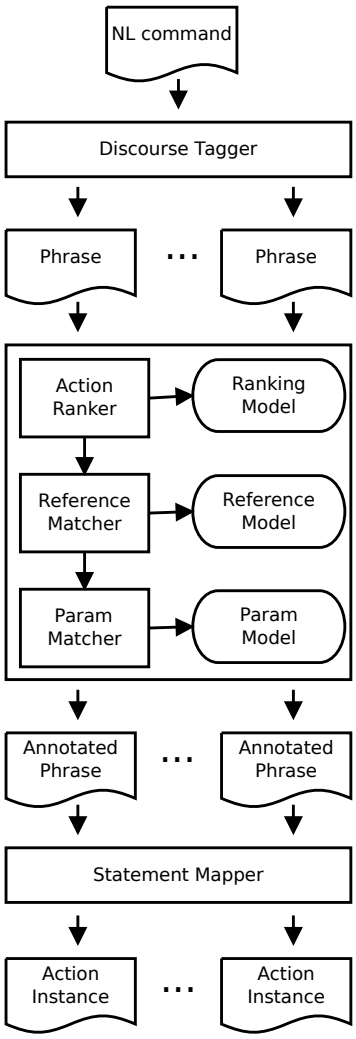

Figure 1: System Components

3.1 Discourse Tagger

The discourse tagger consists of a set of hard-coded rules that are responsible for splitting the command into separate phrases that can be passed to the action ranker independently. The rules are implemented as Python functions that check if a given token can be a split point and assign tags that identify the relationships that hold among phrases being separated. For example, the discourse tagger has a rule that checks if both sides of the *and* token contain tokens that are verbs (i.e. are annotated with the `VERB` part-of-speech tag) and if the condition is satisfied, the tagger splits the command into two phrases and annotates the second phrase with the `AN` tag. Beside the rules for separating sequences of actions, the tagger also contains rules that identify conditional statements and assign `IF`, `DO` and `EL` tags that indicate the condition, consequence and alternative parts. Since while loops oc-

curred in the training set sparsely, we did not introduce tags to annotate them restricting our attention to sequences of actions and conditional statements only.

3.2 Action Ranker

For the purpose of ranking the actions we considered TF-IDF (Spärck Jones, 1972) and Doc2Vec (Le and Mikolov, 2014) document similarity models.[2] We represent the actions by documents that consist of the text collected from the selected fields of the action definitions specified in the `actionkb` file and natural language commands provided in the `mapping` file. The gathered text is lemmatized and stop words are eliminated. Furthermore, the natural language commands are delexicalized by replacing occurrences of named entities and quotations with placeholder tags. In order to determine candidate actions for the given command, we apply the same text preprocessing rules as above and select actions that correspond to the documents that are most similar to the preprocessed command.[3]

We performed 5-fold cross-validation on the training set to select features for our final model. We investigated the models that gather text from: action names (N), action descriptions (D), `provider` fields (P); names and descriptions of action parameters (Par); names and descriptions of action `data` fields (Dat); natural language commands from the `mapping` file (Com). The average Micro F1 scores and their standard deviations across validation folds are reported in Table 1. It may be noticed that results achieved by introduction of action descriptions and `provider` fields to the models surpass the results of the models that consists of the action names only. Conversely, the extension of the NDP (name, description and `provider`) model with the names and descriptions of action parameters (NDPPar) worsens the results. The same holds if we extend the NDP model with action data values (NDPDat). Finally, the feature that improves the results most consists of the aggressively normalized natural language commands from the `mapping` file (Com). This exemplifies that even a small sample of the annotated input data can improve the results significantly. Another interesting observation is that Dov2Vec models perform considerably worse than TF-IDF models in the task regardless of the feature choice. Thus, for our final submission we selected the TF-IDF model that encompasses action names, their descriptions, `provider` fields from `actionkb` and natural language commands from `mapping` (NDPCom).

[2]For training TF-IDF and Doc2Vec models we used Gensim (Řehůřek and Sojka, 2010).

[3]We limit our attention to up to 10 documents that are within 0.05 distance to the most similar candidate.

3.3 Reference Matcher

The reference matcher is responsible for establishing links between the data returned by an action and the parameters of the succeeding actions. The mapping file distributed with the training data is used to populate the set of constraints for the acceptable links. For any two actions that are linked by an occurrence of the `<return*>` tag, the training procedure collects the identifiers of the linked actions and the names of the `data` and `params` fields being connected.

The matching procedure traverses consecutive phrases of the natural language command. If a phrase contains an anaphor, then the reference matcher checks whether the action instances of the antecedent and current phrases belong to the set of constraints learned during training. If the pair of actions belongs to the constraints set, the links between corresponding `data` and `params` fields are established.

3.4 Parameter Matcher

For the purpose of setting parameter values for actions we began with a sequence model based on the learning to search approach (Chang et al., 2015). Unfortunately, due to relatively small training set, the model became highly over-trained and did not prove to be useful for our final submission. Instead, we decided to use a parameter matcher that aligns data types between tokens that occur in the phrase and action parameters. The matcher consists of two components: parameter type and phrase type inducers.

The parameter type inducer restricts data types that can be accepted by action parameters. Data types are constrained on the basis of feature annotations gathered during the data preparation stage (cf. Section 2). The constraints are learned from the mappings file with the following algorithm:
For every phrase in the mapping file, for every action instance of the phrase, for every parameter of the action instance, let C be the set of constraints of the parameter, let S be the set of phrase tokens that match the parameter value:

Model	Metric	N	ND	NDP	NDPPar	NDPDat	NDPCom
TF-IDF	micro-F1	0.0759	0.1079	0.1146	0.0960	0.1099	**0.2609**
	std. dev.	0.0274	0.0249	0.0302	0.0415	0.0288	0.0935
Doc2Vec	micro-F1	0.0049	0.0543	0.0737	0.0683	0.0866	0.2007
	std. dev.	0.0043	0.0186	0.0076	0.0215	0.0098	0.0531

Table 1: Average Micro F1-Score of TF-IDF and Doc2Vec action ranking models.

1. *Add types of tokens in S to C.*
2. *Add named entity tags of tokens in S to C.*
3. *If S is encompassed by quotation signs, add the* `quotation` *type to C.*

The phrase type inducer constraints data types of the phrase by applying the following procedure to every token T:
Let C be the set of constraints of T:

1. *Add type of T to C.*
2. *Add named entity tags of T to C.*
3. *If T is encompassed by a quotation, add the* `quotation` *type to C.*
4. *If T is a reference to the entry E in the user knowledge base, add data types of all non-empty fields of E to C.*

We assume that an action parameter has to be matched, if the intersection of the sets of constraints returned by both inducers is non-empty. Initialization of the parameter value is a 2-step procedure. If the token type belongs to the set of constraints returned by the parameter type inducer, then the raw text of the token is appended to the parameter value. Otherwise, if the token is a reference to the entry in the user knowledge base, the parameter value is populated with the value of an entry field that satisfies the constraints returned by the parameter type inducer.

3.5 Statement Mapper

As in the case of the discourse tagger, the statement mapper consists of a set of deterministic, hard-coded rules implemented in Python that are responsible for converting the phrases annotated with actions assigned by the action ranker, links established by the reference matcher and parameter values set by the parameter matcher to the output format described in the task definition. The relationships among phrases identified by the discourse tagger and encoded as `AN`, `IF`, `DO` and `EL` tags are used by the statement mapper to create JSON objects that represent sequential and conditional execution of action instances in accordance with the task specification.

4 Results

The results of evaluation performed according to the official task criteria are gathered in Table 2. The detailed error analysis requires access to the annotated version of the test set which was not available at the time of writing. Nevertheless, some initial observations can be drawn. The TF-IDF model built from action names, their descriptions, provider names and exemplar phrases seems to be a reasonable baseline for determining the ranking of actions that results in solving of 13 out of 31 scenarios if the parameter values are not considered. On the other hand, the parameter matching strategy requires considerable improvement. We suspect that our per-token strategy leads to the parameter values that are only partially matched, hence do not contribute to the result. Another issue that has to be approached in the future is the problem of propagating data type constraints from parameters of actions that occur in the set of training commands to the parameters of actions for which the training instances are not available.

Criterion	Metric	Value
Individual actions solved ignoring parameter values	precision recall	0.5490 0.7066
Individual actions solved considering parameter values	precision recall	0.0533 0.0533
Scenarios solved ignoring parameter values	accuracy	41.93%
Scenarios solved considering parameter values	accuracy	0%

Table 2: Evaluation results.

References

Saeed Aghaee and Cesare Pautasso. 2014. End-user development of mashups with naturalmash. *Journal of Visual Languages & Computing* 25(4):414–432.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042* .

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to search better than your teacher. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. JMLR Workshop and Conference Proceedings, pages 2058–2066. http://jmlr.org/proceedings/papers/v37/changb15.pdf.

Joëlle Coutaz, Alexandre Demeure, Sybille Caffiau, and James L Crowley. 2014. Early lessons from the development of SPOK, an end-user development environment for smart homes. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, pages 895–902.

Edsger W Dijkstra. 1979. On the foolishness of "natural language programming". In *Program Construction*, Springer, pages 51–53.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 363–370.

Marek Kubis, Paweł Skórzewski, and Tomasz Ziętkiewicz. 2017. EUDAMU System Components Data Flow. https://bitbucket.org/mapato/eudamu/wiki/DataFlow.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. pages 1188–1196. http://jmlr.org/proceedings/papers/v32/le14.html.

Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-user development: An emerging paradigm. In *End user development*, Springer, pages 1–8.

Greg Little, Tessa A Lau, Allen Cypher, James Lin, Eben M Haber, and Eser Kandogan. 2007. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, pages 943–946.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

Fabio Paternò. 2013. End user development: Survey of an emerging field for empowering people. *ISRN Software Engineering* 2013.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, pages 45–50. `http://is.muni.cz/publication/884893/en`.

Juliano Sales, Siegfried Handschuh, and André Freitas. 2017. SemEval-2017 Task 11: End-User Development using Natural Language. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics, Vancouver, Canada, pages 554–562. http://www.aclweb.org/anthology/S17-2092.

Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28(1):11–21.