Context Classification in Dialog-Based Interaction

Alexander Wachtel, Felix Eurich, Dominik Fuchß, and Walter F. Tichy
Karlsruhe Institute of Technology
Chair for Programming Systems
Karlsruhe, Germany
alexander.wachtel@kit.edu, felix.eurich@student.kit.edu,
dominik.fuchss@student.kit.edu, walter.tichy@kit.edu

***Abstract*—A difficulty in processing of the natural language is recognizing the context of a statement. However, since this contains implicit knowledge which we unconsciously use in our formulations, it is necessary to assign the context for correct interpretation. A context in our system is managed by a service that interprets and processes input and provides feedback to the end user. In this paper, we present a solution how an end user input can be assigned to such a service. For this purpose, we score the end user inputs by the system with an ensemble of 6 different classifiers that consider semantics as well as syntax. The system learns the user's input at run time and adapts his enunciation step by step. During the evaluation, the system was able to classified 87% of the user statements to the correct service. Far from perfect, this research might lead to fundamental changes in computer use.**

***Index Terms*—Context; Natural Language User Interfaces; Human Computer Interaction; Classification.**

I. Introduction

We aim for a major breakthrough by making computers programmable in ordinary, unrestricted, written or spoken language. Advanced AI and logic techniques will enable laypersons to instruct programmable devices, without having to learn a programming language. This research might lead to fundamental changes in computer use. Rather than merely consuming software, users of the ever-increasing variety of digital devices and software building blocks could develop their own programs, potentially leading to novel, highly personalized, and plentiful solutions.

Due to the natural language, there are several user inputs $i_2; \ldots ; i_N$ that are similar to i_1 and also map to the known action a_1. There is also a problem on the classification of required system action or skill. The match of the user input i_1 to system action a_1 is unique until the function of the system could be clearly divided by the syntax of the user input. In case of object-oriented programming, the user input can be referenced to the Algorithm skill or Class interpretation skill. They have similar inputs but different actions on execution. For this reason, it is not enough to check the language on the syntactical level. Furthermore, the syntax matching should also be extended by the interpretation on the meta level. These methods should combine syntax, semantic and context classification. In our next paper, we will present the classification with several classifiers. The result of the individual classifiers is summed and forms.

A. Statement of a problem

Our research is to improve the interaction between humans and machines and enable the end user to instruct programmable devices, without having to learn a programming language. Therefore, the system (i) enables end users to give instructions step-by-step, to avoid the complexity in full descriptions and give directly feedback of success (ii) creates an abstract meta model for user input during the linguistic analysis and (iii) independently interprets the meta model to code sequences that contain loops, conditionals, and statements. The context then places the recognized program component in the history. In this way, an algorithm is generated in an interactive process (See Figure 1).

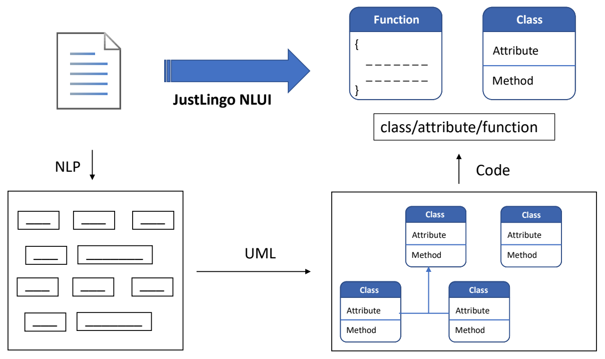


Fig. 1. JustLingo NLUI

We implemented several services: calculations, data analysis, algorithm interpreter, and entity recognizer. After, we face to a classification problem of the required system services due to the ambiguity of the natural language. The match of the user input $input_1$ to system action $action_1$ is unique until the function of the system could be clearly divided by the syntax of the user input. In case of entity recognizer, the user input can be referenced to the algorithm service or class interpretation service. These services have similar inputs but different actions on execution. For this reason, it is not enough to check the language on the syntactical level. These methods should combine syntax, semantic and context classification. For that we use several different classifiers. The result of the individual classifiers is summed and forms the overall similarity of the input for the specific service.

B. Services in Short

a) Calculations: In 2015, Wachtel et. al [1] enabled the dialog-based interaction with end users in natural language and calculation of simple arithmetical tasks within spreadsheets.

Furthermore, the system resolves references to previous results and allows the construction of complex expressions step-by-step. The architecture consists of a user interface responsible for human interaction, as well as a natural language understanding and a dialog management unit.

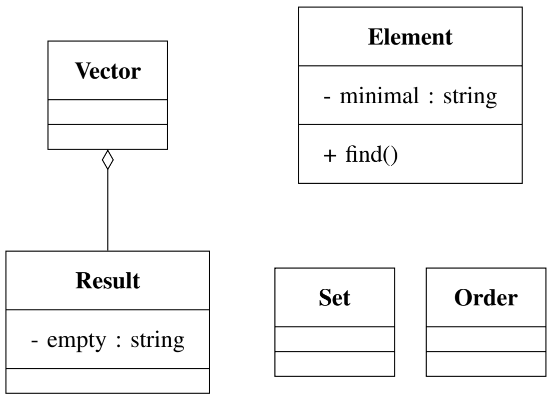
b) Entity & Structure Recognition: In general, classes represent entities from the real world, attributes specify different properties of an entity, and methods are functions or algorithms that allow to manipulate these attributes [2] .

$$Sorting\ a\ sequence\ of\ n\ numbers\ (a_1, ..., a_n)$$

$$\downarrow$$

End user input in natural language:
The result is a vector. Initially it is empty. Find the minimal element of the set and append it to the vector. Remove the element form the set. Then, repeatedly find the minimum of the remaining elements and move them to the result in order, until there are no more elements in the set.

$$\downarrow$$



$$\downarrow$$

```
namespace Name
{
  class Result : Vector {
    private string empty;
    public Result(string newempty) {
        this.empty = newempty;
    }
  }
  class Element {
    private string minimal;
    public Element(string newminimal) {[...]}
    public void find() { }
  } [...]
}
```

c) Algorithms: End users describe algorithms (cf. [3]) in their natural language and get a valid output by the dialog system for given description, e.g., selection sort of a list. In 2018, Wachtel et. al [4] presented the natural language processing algorithm service that recognizes and implements methods described by a user. The functionality is aimed at users with no programming knowledge, as the system enables simple routines to be programmed without prior knowledge. This makes it easier for users to get started with programming. The algorithm interpreter provides the following code for the example from the last section:

Algorithm 1 Pseudo code of selection sort.

```
1: procedure SelectionSort(input : list of numbers)
2:     result ← []
3:     while IsNotEmpty(input) do
4:         n ← Length(input) - 1
5:         tmpMin ← 0
6:         for $i$ ← 0 → n do
7:             if input[i] < input[tmpMin] then
8:                 $tmpMin \leftarrow i$
9:         Append input[tmpMin] to result.
10:        Remove at index tmpMin from input.
```

$$\downarrow$$

$$A\ permutation\ (b_1, ..., b_n)\ with\ b_1 \leq ... \leq b_n$$

II. Context Classification

In our system context is represented by services. A service is responsible for processing inputs that are in its context (See Figure 2). To assign whether an input belongs to the context of the service, we use its input history. This grows continuously as the system is used.

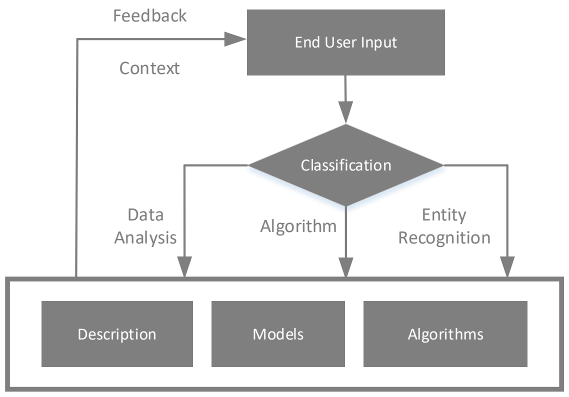


Fig. 2. Classification of similar end user inputs

As a result, the accuracy of the classification is constantly increasing. For classification, we use bagging in combination with seven classifiers from DKPro Similarity [5]. Four of them consider the lexical similarity of sentences:

- NGram compares the n-element parts of two propositions, we use n = 2.
- CosinusSimilarity forms vectors from the propositions and then calculates the cosinus as a measure of similarity. The length of the vector corresponds to the number of different words of the two propositions. The vector of a sentence has a 1 at the places, if it contains the word, otherwise a 0.
- GreedyStringTrilling considers the longest contiguous part that both propositions have in common. The minimum overlap length is two words.
- MongeElkan considers how well the words of one sentence fit to the words of another sentence on average. Basis for the comparison of two words is the Jaccard coefficient.

For the analysis of the semantic similarity we use a concept vector based measure in combination with concept representation of Wiktionary and Wordnet [6].

The basic procedure is as follows: (1) The history of a service is used to evaluate the relevance of the individual classifiers. (2) The average match of new entries with the history of a service is weighted and added according to the relevance of the respective classifier. (3) The calculated score of the services is passed on to the dialog system. Furthermore, the system is designed so that services and classifiers can be added easily.

Algorithm 2 Train classifiers for a service.

```
 1: procedure TRAINCLASSIFIER(srv : service)
 2:     for each classifier c do
 3:         averageScores ← []
 4:         for sentence $s$ in past inputs of $srv$ do
 5:             // Get all past sentences without $s$
 6:             other ← $History(srv) \setminus \{s\}$
 7:             scores ← []
 8:             for each sentence $o \in other$ do
 9:                 score ← c.getSimilarity(s, o)
10:                 scores.add(score)
11:             averageScores.add(score average)
12:         map ← srv.classifierToAverageScore
13:         map.put(c, averageScores)
14:     CalcWeights(srv)
```

A. Preprocessing

The sentences are pre-processed before being processed by the ensemble. Upper-case letters are replaced by lower-case letters, stop words and punctuation marks are removed. The aim is to simplify the input and to limit it to the essential, which is especially important for lexical analysis.

B. Training

The classifiers are trained per context and receive a weighting that reflects how helpful they are for the recognition of relevant sentences in this context (see Algorithm 2).

In order to estimate the relevance and thus the weight of the individual classifiers, their proportion of the summed average score for the respective service is calculated.

After the training, a service learns continuously. The dialog system, which uses the result of the classification to process the input, provides our system with the information if the classification was correct or, if not, which context the user meant with his input. In this way, sentences can be added to the corresponding service and thus on the one hand the weights of the classifiers can be updated (see Algorithm 3) and on the other hand the new sentence can be used for future similarity analysis.

Algorithm 3 Learn new input by sentence and service.

```
 1: procedure LEARNSENTENCE( s, srv )
 2:     // Get all past sentences without $s$
 3:     other ← $History(srv) \setminus \{s\}$
 4:     for each classifier $c$ do
 5:         scores ← []
 6:         for each sentence $s$ in History(srv) do
 7:             score ← c.getSimilarity(s, o)
 8:             scores.add(score)
 9:         map ← srv.classifierToAverageScores
10:         map[c].add(average of scores)
11:     CalcWeights(srv)
```

C. Classification

Algorithm 4 describes the procedure for calculating the similarity assignment of a new input. All services are iterated and a value for the match with the history of the services is calculated for the various elements of the ensemble. This value is then summed weighted and results in the total score of the service. The assignment of the services to the calculated scores is then returned and and is used by the dialog system for further processing.

D. Performance

Due to the fact that in our classification we always consider all historical inputs of the service for each classifier per service, the optimization of the performance plays an important role. After all, the user should not have to wait when entering a service. Two mechanisms were used to achieve this goal: asynchronous calls and prioritization.

The calculation of the relevance of a service is performed asynchronously. For each service, one thread is started for calculating the relevance score. Within this thread, the user input is now processed as described in Algorithm 4. The relevance of a service is calculated iterative and the intermediate result can be retrieved at any time. In this way it is possible to return the intermediate result or the final result directly after a freely definable minimum calculation time.

Algorithm 4 Calculate similarity assignment for a new input.

```
procedure CalcSimilarity( input )
    serviceToScore ← empty map
    for each service parallel do
        relevance ← 0
        sort sentence of service according
        to their relevance for the
        classification result in past
        for classifier c do
            scores ← empty list
            for each sentence $s$ in the past do
                score ← c.getScore(input, s)
                if score $\geq$ threshold then
                    s.increaseRelevance()
                scores.add(score)
            // Calculate aggregate
            aggr ← max(scores) - sd(scores)
            // Update relevance
            relevance ← relevance
                + (aggr $*$ srv.getWeight(c))
        serviceToScore.add(srv, relevance)
    return serviceToScore
```

The sentences of a service used for classification are prioritized processed, so that even the intermediate results have a high expressiveness. Sentences that were often helpful for classification in the past are considered first. A sentence is regarded as helpful if it has achieved a higher score than a defined threshold from a classifier in a comparison with a new sentence (for us 0.5). For each classified input, the priority of one or more records in the history increases. This procedure results in the system adapting to the user's mode of expression and producing better results faster and faster over time.

Of course, at some point it will no longer be possible to consider all historical entries, but in the order of magnitude in which we move, this can be done without any problems. Our goal is not to provide an all-encompassing or large amount of training data per service. Instead, we use a compact training size and then adapt to the user during the use of the system. For other application areas it would be conceivable to define a maximum number of historical inputs to be considered. The important and helpful sentences would still be available, thanks to the constant prioritisation.

III. Evaluation

We took a data set of 200 samples (50 from each book) for evaluation from different books of the regarding language domain: *Basic Engineering Mathematics* [7] (e.g.”The four basic arithmetic operators are add, subtract, multiply and divide.”), *The State of the Art in End-User Software Engineering* [8] (e.g.Researchers have also extended their focus from perfective activities to design, including work on requirements, specifications, and reuse.””), *Data Analysis Using SQL and Excel* [9] (e.g.”The definition of the tables, the columns, and the relationships among them constitute the data model for the database.”), and *Introduction to Algorithms* [3] (e.g.”Typically, we use the loop invariant along with the condition that caused the loop to terminate.”). We split the samples from each set randomly in 80% train and 20 % test data. First, we trained the classifier with train data set examples and evaluated the classification with test data set.

TABLE I
Confusion matrix

		Actual class			
		Struct.	**Calc.**	**Data**	**Algo.**
Predicted class	**Structure**	46	4	0	0
	Calculation	3	44	3	0
	Data	1	6	43	0
	Algorithms	3	3	3	41

Although our data situation is still small, the results show that the classification seems possible. The possibility for the user to manually define the service in the dialog window also allows further training data to be collected.

TABLE II
Accuracy, Precision, Recall and F1 Score

	Acc	**Prec**	**Rec**	**F1**
Structure	94,5%	0.92	0.87	0.89
Calculation	90,5%	0.88	0.77	0.82
Data	93,5%	0.86	0.88	0.87
Algorithms	95,5%	0.82	1.0	0.90

IV. Related Work

In 2018, Howard proposed Universal Language ModelFine-tuning (ULMFiT) for text classification [10]. It is an effective transfer learning method that can be applied to any task in NLP. Also, Young presents an overview about a variety of model designs and methods have blossomed in the context of natural language processing [11]. In general, the idea of programming in natural language was first proposed by Sammet in 1966 [12], but enormous difficulties have resulted in disappointingly slow progress. In recent years, significant advances in natural language techniques have been made, leading, for instance, to IBM's Watson [13] computer winning against the two Jeopardy! world champions, Apple's Siri routinely answering wide-ranging, spoken queries, and automated translation services such as Google's becoming usable [14], [15]. In 1979, Ballard et al. [16]–[18] introduced their Natural Language Computer (NLC) that enables users to program simple arithmetic calculations using natural language. Although NLC resolves references as well, there is no dialog system. Metafor introduced by Liu et al. [19] has a different orientation. Based on user stories the system tries to derive program structures to support software design. Also, NLP2Code [20] enables developers to request for code snippets from the Internet, e.g. Stack Overflow, and integrate these snippets directly into the source code editor. This solution works if the developer needs matches the already existing code snippets.

Paternò [21] introduces the motivations behind end user programming defined by Liberman [22] and discusses its basic concepts, and reviews the current state of art. Various approaches are discussed and classified in terms of their main features and the technologies and platforms for which they have been developed. In 2006, Myers [23] provides an overview of the research in the area of End-User Programming. As he summarized, many systems for End User Development have already been realized [24], [25], [26]. In 2008, Dorner [27] describes and classifies End User Development approaches taken from the literature, which are suitable approaches for different groups of end users. Such environments enable differently serviced end users to perform system adaptations on their own. Begel [28] introduces voice recognition to the software development process. It uses program analysis to code in natural language, thereby enabling the creation of a program editor that supports voice-based programming.

V. Conclusion and Future Work

In this paper, we presented classification of the end user input to the existing services. Without classification, it is not possible to interact with the system and be able to provide inputs across all services. During the implementation, we used all services one by one. Real-world scenario is to use all services at the same time and also allow inputs across services without interruption of the natural dialog flow provided by our system.

However, programming in natural language remains an open challenge [15]. First, we will run classification training with more data on each data set. Regarding the books, we can achieve training sets of 140K samples. Since natural language is ambiguous and our prototype supports more functionality, we face to a new problem. There are several user inputs $i_2, \ldots, i_N$ that are similar to i_1 and also map to the known action a_1. To avoid that, the system needs information on the context end user is talking about.

Ordinary, natural language would enable almost anyone to program and would thus cause a fundamental shift in the way computers are used. Rather than being a mere consumer of programs written by others, each user could write his or her own programs [29].

References

[1] A. Wachtel, "Initial implementation of natural language turn-based dialog system," *International Conference on Intelligent Human Computer Interaction (IHCI)*, December 2015.

[2] A. Wachtel, D. Fuchß, M. Przybylla, and W. F. Tichy, "Natural Language Data Queries on Multiple Heterogenous Data Sources," *The Seventh International Symposium on End-User Development (IS-EUD)*, July 2019.

[3] T. H. Cormen, *Introduction to algorithms*. MIT Press, 2009.

[4] A. Wachtel, F. Eurich, and W. F. Tichy, "Programming In Natural Language Building Algorithms From Human Descriptions," *The Eleventh International Conference on Advances in Computer-Human Interactions*, March 2018.

[5] D. Bär, T. Zesch, and I. Gurevych, "Dkpro similarity: An open source framework for text similarity," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, August 2013.

[6] T. Zesch, C. Müller, and I. Gurevych, "Using wiktionary for computing semantic relatedness." in *AAAI*, vol. 8, 2008, pp. 861–866.

[7] J. Bird, *Basic Engineering Mathematics*. Elsevier, 2010.

[8] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, J. Lawrence, B. Myers, M. B. R. G. Rothrmel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," 2011.

[9] G. S. Linoff, "Data analysis using sql and excel." WILEY, 2008.

[10] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018.

[11] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," in *Cornell University*, 2018.

[12] J. E. Sammet, "The use of english as a programming language," *Communications of the ACM*, vol. 9, no. 3, 1966.

[13] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager *et al.*, "Building watson: An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[14] H. Liu and H. Lieberman, "Toward a programmatic semantics of natural language," in *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE, 2004, pp. 281–282.

[15] C. L. Ortiz, "The road to natural conversational speech interfaces," *IEEE Internet Computing*, vol. 18, no. 2, pp. 74–78, 2014.

[16] B. W. Ballard and A. W. Biermann, "Programming in natural language: Nlc as a prototype," in *Proceedings of the 1979 annual conference*. ACM, 1979.

[17] A. W. Biermann and B. W. Ballard, "Toward natural language computation," *Computational Linguistics*, vol. 6, no. 2, pp. 71–86, 1980.

[18] A. W. Biermann, B. W. Ballard, and A. H. Sigmon, "An experimental study of natural language programming," *International journal of man-machine studies*, vol. 18, no. 1, pp. 71–87, 1983.

[19] H. Liu and H. Lieberman, "Metafor: visualizing stories as code," in *Proceedings of the 10th international conference on Intelligent user interfaces*. ACM, 2005, pp. 305–307.

[20] B. A. Campbell and C. Treude, "NLP2Code: Code Snippet Content Assist via Natural Language Tasks," *The International Conference on Software Maintenance and Evolution (ICSME)*, August 2017.

[21] F. Paternò, "End user development: Survey of an emerging field for empowering people," *ISRN Software Engineering*, vol. 2013, 2013.

[22] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm," in *End user development*. Springer, 2006, pp. 1–8.

[23] B. A. Myers, A. J. Ko, and M. M. Burnett, "Invited research overview: end-user programming," in *CHI'06 extended abstracts on Human factors in computing systems*. ACM, 2006, pp. 75–80.

[24] A. J. Ko and B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behavior," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, pp. 151–158.

[25] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Communications of the ACM*, vol. 55, no. 8, pp. 97–105, 2012.

[26] A. Cypher and D. C. Halbert, *Watch what I do: programming by demonstration*. MIT press, 1993.

[27] M. Spahn, C. Dörner, and V. Wulf, "End user development: Approaches towards a flexible software design." in *ECIS*, 2008, pp. 303–314.

[28] A. B. Begel, *Spoken language support for software development*. University of California, Berkeley, 2005.

[29] W. F. Tichy, M. Landhäußer, and S. J. Körner, "Universal Programmability - How AI Can Help. Artificial Intelligence Synergies in Software Engineering," May 2013.