Programming in Natural Language: Building Algorithms from Human Descriptions

Alexander Wachtel, Felix Eurich, Walter F. Tichy

Karlsruhe Institute of Technology
Chair for Programming Systems Prof. Walter F. Tichy
Am Fasanengarten 5, 76131 Karlsruhe, Germany
Email: alexander.wachtel@kit.edu, felix.eurich@student.kit.edu, walter.tichy@kit.edu

***Abstract*—Our work is where the Software Engineering meets the Human Computer Interaction and the End User Programming to aim for a major breakthrough by making machines programmable in ordinary and unrestricted language. In this paper, we provide a solution on how new algorithms can be recognized and learned from human descriptions. Our focus 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. In a test-driven development, we created a platform that allows users to manipulate spreadsheet data by using natural 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. One of the result can be the code sequence for algorithm, like well-known selection sort. We present a series of ontology structures for matching instructions to declare variables, loop, make decisions, etc. Furthermore, our system asks clarification questions when the human user is ambiguous. During the evaluation, 11 undergraduate students were asked to solve tasks by using natural language, and describe algorithms in three classes of complexity. Overall, the system was able to transform 60% of the user statements into code. Far from perfect, this research might lead to fundamental changes in computer use. Rather than merely consuming software, end users of the ever-increasing variety of digital devices could develop their own programs, potentially leading to novel, highly personalized, and plentiful solutions.**

Keywords–Natural Language Processing; End User Programming; Natural Language Interfaces; Human Computer Interaction; Programming In Natural Language; Dialog Systems.

I. INTRODUCTION

Since their invention, digital computers have been programmed using specialized, artificial notations, called programming languages. Programming requires years of training. However, only a tiny fraction of human computer users can actually work with those notations. With natural language and end-user development methods, programming would become available to everyone and enable end-users to program their systems or extend them without any knowledge of programming languages. Programming languages assist with repetitive tasks that involve use of loops and conditionals. This is what is often challenging for users. Our vision should enable users to describe algorithms in their natural language and provides a valid output by the dialog system for a given description, e.g., selection sort of a set (See Algorithm 1). This vision forms the basis for our natural language user interface [1]. Already in 1987, Tichy discussed that Artificial Intelligence (AI) techniques are useful for software engineering, pointing out the potential of natural language processing [2] and natural-language help systems [3].

According to Liberman [4], the main question in the End User Development area of research is, how to allow non-programming users who have no access to source code, to program a computer system or extend the functionality of an existing system. In our prototype, we decided to address spreadsheets for several reasons:

- a lot of open data available, e.g. Eurostat [5] provides statistics for European Union that allow comparisons.
- well-known and well-distributed: Microsoft [6] announced the distribution of Office with 1.2 billion users worldwide. Furthermore, for humans the data in a table is easy to understand, complete and manipulate.

In general, spreadsheets have been used for at least 7000 years [7]. Myers [8] and Scaffidi [9] compared the number of end users and professional programmers in the United States. Nearly 90 million people use computers at work and 50 million of them use spreadsheets. In a self-assessment, 12 million considered themselves as programmers, but only 3 million people are professional programmers. The created spreadsheets are not only the traditional tabular representation of relational data that convey information space efficiently, but also allow a continuous revision and formula-based data manipulation. It is estimated that each year hundreds of millions of spreadsheets are created [10]. Our system consists of two main components: a user interface that gets natural language expressions from the user and a linguistic analysis framework. These components are explained below. This paper is structured as follows. Section II gives the overview of the current research and describes our goals. This is followed by the Section III, which presents the linguistic analysis that generates the meta model for given descriptions. In Section IV, the interpreter transforms the meta model to the control flows, conditional and loop statements. Section V evaluates the prototype and describes the setup and the results of a user study. Section VI presents related work in the research areas of programming in natural language, End User Programming and natural language dialog systems. Finally, Section VII presents a conclusion of our topic and future work.

 ISBN: 978-1-61208-616-3

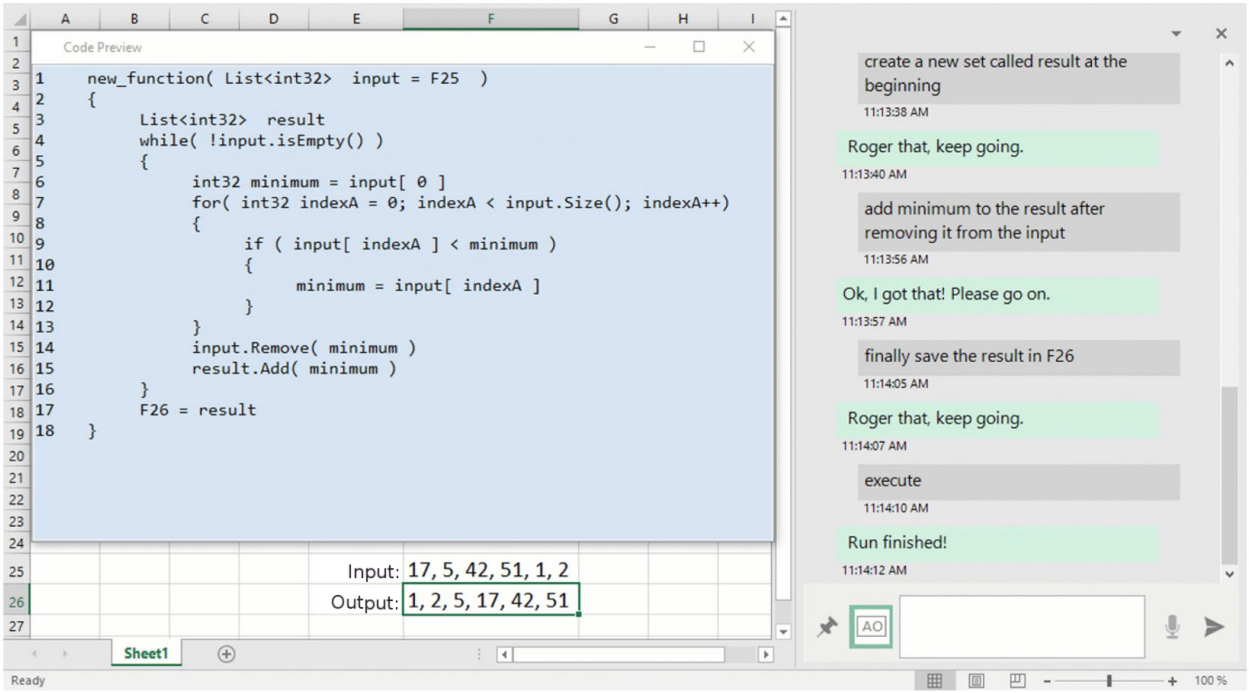

Figure 1. End user instructions of an algorithm are transformed to the code sequences and executed on a given set of numbers

II. CURRENT WORK AND RESEARCH GOALS

Our work is based on a dialog component implemented by [11] in 2015. It enables users to interact with our system and manipulate spreadsheet data via natural language. In early 2016, the system has been extended with an active ontology [1][12]. The idea of active ontology was first presented by Guzzoni [13]. In general, an ontology is a formal representation of knowledge. By adding a rule evaluation system, a fact store and sensor nodes to an ontology it becomes an execution environment rather than just a formal representation of knowledge. In late 2016, the natural language dialog system has been extended with a machine learning component, called Interactive Spreadsheet Processing Module (ISPM) [14]. It is an active dialog management system that uses machine learning techniques for context interpretation within spreadsheets and connects natural language to the data in the spreadsheets. First, the rows of a spreadsheet are divided into different classes and the table's schema is made searchable for the dialog system. In the case of a user input, it searches for headers, data values from the table and key phrases for operations. Implicit cell references like "people of age 18" are then resolved to explicit references using the schema. Using the ISPM, end users are able to search for values in the schema of the table and to address the data in spreadsheets implicitly, e.g., *What is the average age of people in group A?*. Furthermore, it enables them to select and sort the spreadsheet data by using natural language for end user software engineering, to overcome the present bottleneck of professional developers. In December 2017, we presented an overview about our current work in [15]. Therefore, we provided more details for the achieved design and implementation steps of our prototype.

In our current work, users describe algorithms (cf. [16]) in their natural language and get a valid output by the dialog system for a given description, e.g., selection sort of a set (See Figure 1). 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 system also illustrates the relationship between a natural statement and its code representation (See Figure 1), so it can also help to understand and learn a programming language.

$$Task: To\ sort\ a\ sequence\ of\ n\ numbers\ (a_1, ..., a_n)$$

↓

User Input: 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.

↓

Algorithm 1 Pseudo code of selection sort.

```
1:  procedure SELECTIONSORT( input as a set of numbers )
2:      result ← empty set
3:      while input IsNotEmpty do
4:          n ← length (input) - 1
5:          tmpMin ← 0
6:          for i ← 0 → n do
7:              if input[i] < input[tmpMin] then
8:                  tmpMin ← i
9:          Add input[tmpMin] at the end of result.
10:         Remove element at index tmpMin from input.
```

↓

$$Output: A\ permutation\ (b_1, b_2, ..., b_n)\ with\ b_1 \leq ... \leq b_n$$

III. Linguistic Analysis

In early 2017, we have done first steps on loops and conditionals [1]. In this work, we extend these active ontologies [1] that represent knowledge about the structure of natural language in the context of programming instructions and implementing a meta model layer, our system recognizes various programming concepts in user input and converts them into a meta language. The result of the analysis is a tree structure consisting of objects, whereby an object represents a recognized concept. These trees are built during the analysis from bottom up. The object structure contains three parts: whileIndicator, loopAction, and condition. The following sections describe the different steps on how our system creates shown WhileLoop object. Overall, it identifies the routines of the supported concepts, like statements, conditionals and loops, and provides functional overview of the linguistic analysis.

A. Reference Detection

The first step of the analysis is reference resolution. End users can provide the reference to variable as an individual pseudonym like the word ”minimum” in *use 5 as minimum*. Furthermore, an implicit reference can be used such as *initialize it with 5* (See Figure 2). Dotted lines indicate sensor nodes which react if one of the attached word or word class [17] was detected in the input. A red mark around a node points to a key node which occurrence is essential for the parent node to trigger. The two inner node types: collect and select nodes, are marked with ”C” and ”S”, respectively. The different types of nodes have different prerequisites for triggering. Leaf nodes unleash when a particular word is found in the input. Collecting nodes trigger when all required child nodes have been triggered and selection nodes trigger as soon as at least one child node triggers. When triggered, the recognized concept is passed on to the parent node as a new object [1].

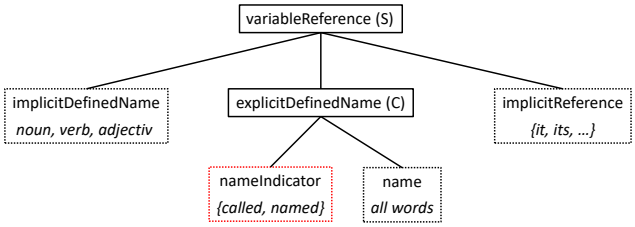

Figure 2. Ontology structure for variable references.

Detecting references to a quantity builds on the previous structure. Decisive for the recognition are descriptions of characteristics (`setIndicator`) which are linked to a variable reference. The simplest indicator is the occurrence of a size reference like the word *length*. In connection with a variable reference the presence of a reference to a quantity can be assumed. Another indicator is the detection of index referral. It’s based on the identification of the explicit index as a constant number or reference (See Figure 3).

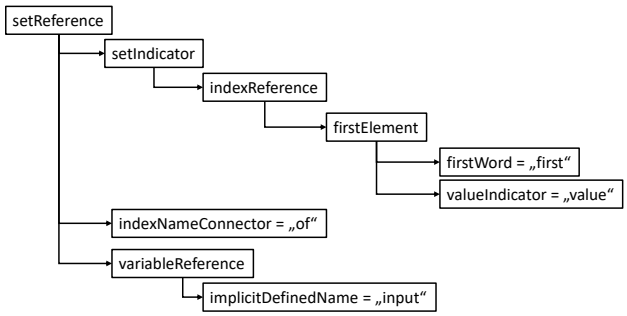

Figure 3. Result of ”set the first element of the input to 0”.

B. Statements

Built on the of reference recognition, declarations and actions are recognized by the linguistic analysis.

1) Declarations: In general, a new variable is defined explicitly by its name and a type. It is important to know that the type is not describing the data type, but the type of the variable, i.e., whether it’s a simple variable or a set. In order to deal with formulations of the form ”create a new set”, the specification of a name is optional. If no name is found for the new variable, the system assigns one.

2) Actions: The current version of the system supports three types of actions, namely assignments, remove and add operations. All actions have the commonality that they have a direction. This is determined by the preposition and the verb used. On the basis of this information, the analysis can indicate which reference is the target and which is the parameter of the action. The set-up is the same for all actions and should be displayed on the basis of the assignment A possible entry for an assignment is the following ”take the first value from the input as minimum”. By the preposition ”as” in combination with the verb ”take” the analysis assumes that the target of the allocation is on the right side of the preposition. Therefore, the tree in Figure 4 is the output of the analysis. The other actions are also recognized according to this procedure.

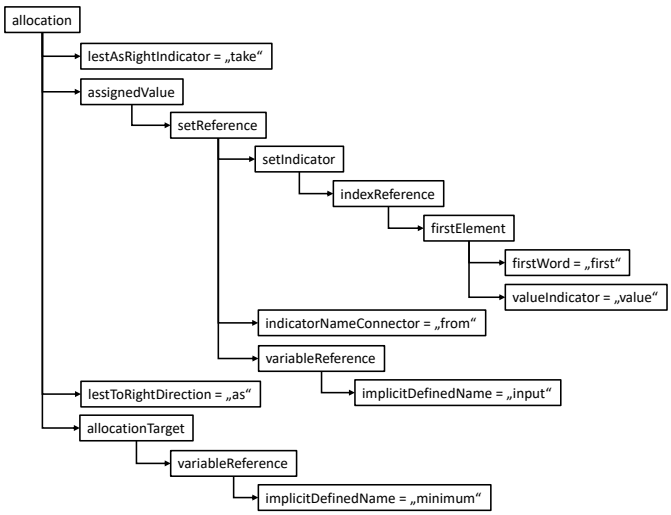

Figure 4. Result of the node `allocation` after the analysis of the entry ”take the first value from the input as minimum”.

 ISBN: 978-1-61208-616-3

C. Conditionals

Control structures such as looping and branching are essential elements of algorithms. By considering such concepts in the linguistic analysis, more complex algorithms like selection sort can be implemented with the system. As some control structures require the recognition of conditions, the logic used for this purpose should be briefly presented here. In general, a condition defies a property of a reference. Depending on the type of reference, different conditions can be set. In this work, a condition can be set for a simple value, a quantity of values and a quantity reference. Figure 5 shows the classification of the conditions. In a first subdivision, a distinction is made between conditions which relate to a single reference (`singleValueCondition`) and the composite condition (`multipleValueCondition`) which is the combination of several conditions by a conjunction such as ”and” in ”x is less than 3 and y is equal to 5”. Furthermore, the individual conditions differentiate between statements that assign a condition to a value (`singleCondition`) and those with several premises for a value (`conditionChain`). The child nodes of `singleCondition` represent the three supported types of a condition. `limitedValueCondition` recognizes statements that compare two values. Conditions which relate to a quantity, e.g., ”If the result is empty” are identified by `setConstrain`. `setCondition` recognizes the requirements of a property for certain elements of a set, such as ”if all elements of the result are greater than 4”.

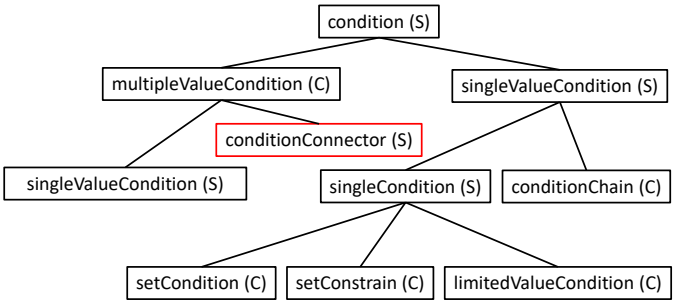

Figure 5. Ontology structure for the different conditions.

1) Single Value Condition: Assuming the input is ”check if the result is less than 5”. It is obvious that the user wants to check whether the value of the field ”result” is less than the constant value 5. Analogously, it would also be conceivable to require equivalence or greater than 5. Of course, negated expressions such as ”when the result at least 3 is not equal to minimum” must be recognized. For this purpose, a negative prefix, like ”not”, is searched for in the input. The `constantValueOrReference` node is represented a all previously discussed ways to reference a single value or a variable. Because of its identical structure, the `containRelation` is also part of this type of condition. The only difference is that a keyword such as ”contain” is searched instead of the requested comparison operator.

2) Quantifier Condition: With these kind of conditions, formulations such as ”check if all elements in the input are greater than 5” or ”if any element of the input is smaller than 0” are correctly recognized Since the set reference to be checked is identified by the `setCondition` node, only the value has to be found in the subtree of `singleValueConditionWithoutLoop` instead of the value and the reference to be checked for the comparison. The already known structure of `possibleRelation` and `valueLimit` remains unchanged.

3) Constraint Condition: Entries such as ”if the input contains any element” are handled by the `setConstrain` node. It is thus possible to check whether a quantity reference contains elements or not.

4) Condition Chain: The condition chain is a special compound condition, which requires several properties from a single reference. The decisive factor here is that only the first requirement has an explicit reference to the field to be examined. A possible input would be ”if x is less than 5 or equal to 10”. Instead of whole conditions, operators can also be linked. An example of this is the input ”if x is smaller or equal to 4”. In contrast to the linking of several conditions, only the last operator has an explicit reference to the limit.

D. Loops

In order to repeat an action, the user can link it to a condition or specify that it should be executed for each element of a set. Hence the analysis detects both `for` and `while` loops. The result of the ontology for a recognized `for` loop can be seen in Figure 6. Similarly, the result for the input ”repeat all steps until the input is empty” will be computed.

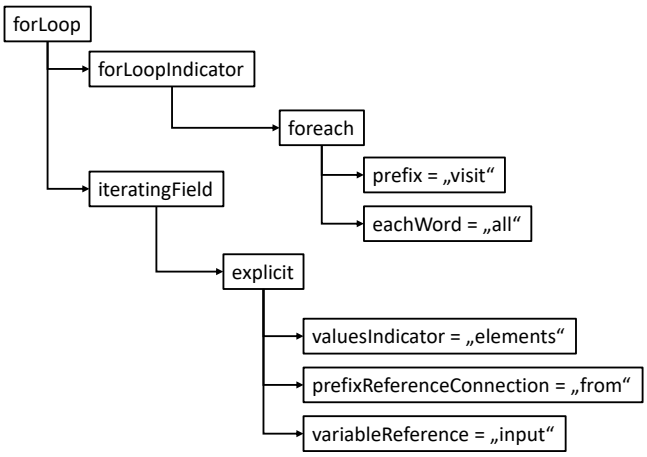

Figure 6. Result of the node `forLoop` after the analysis of the entry ”visit all elements from the input”.

E. Branches

Analogous to the branching in programming, the input is checked for conditional actions and possible alternatives. In fact, the system is able to trace and identify such user input like ”if there is a value less than the minimum in the input use it as minimum”.

F. Temporal Actions

Temporal actions are actions with an individual desired location. Both explicit position data and the definition of a temporal relationship to another action are taken into account during the analysis. As an explicit position, the start and end of the description is currently supported such as ”at the beginning initialize the output with 0”. The assignment of a

 ISBN: 978-1-61208-616-3

temporal relationship requires, in addition to the action to be placed (`action`), the specification of a conjunction (after, while, before) and an anchor point (`anchor`). As an action or anchor point, the analysis allows both action descriptions, as well as concrete actions or references to earlier instructions. To identify the described component, the analysis looks for matching verbs and adds the corresponding section in the input as a description. It does not matter which action is already present in the program sequence and which is to be rearranged. This task is done by the interpreter. A possible entry for a temporal action would be ”add minimum to the result after removing it from the input”.

IV. INTERPRETER

In the following, the further processing of the object structure outputted by the ontology will be presented. The abstract process is shown in Figure 7. The successive processing steps are now to be considered in more detail.

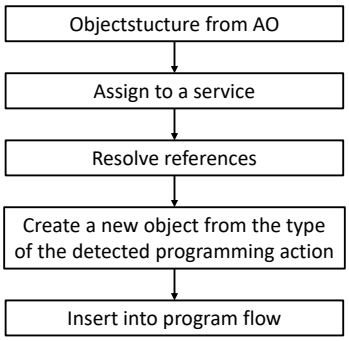

Figure 7. Abstract process in the Interpreter.

A. Interpreter Services

The transformation of the recognized concept was implemented with a micro service architecture. Every recognized programming concept has its own service, which knows what information is necessary for the transformation. In Figure 7, the result of the ontology is passed to the appropriate service on the basis of the detected action. The purpose of the services is to transform the existing information into an executable action. Regardless of the recognized statement, the main task is to resolve the specified references. After that they have to be linked to the action. For this purpose, a new object of the type of the action is generated and the resolved references are used (Section IV-D).

B. Reference Resolving

Similar to the organization of the detected actions, the references are also assigned to a service according to the recognized type. From the description of the linguistic analysis, three types are possible: constants, quantity references, and simple variable references. When searching for references, different information can be provided. Basically, the name of the reference or the indication that it is an implicit reference and the allowed field type must be given. Optionally, the required data type can be transferred. This is determined by the datatype of the first resolved reference. In the event of a directed action, this takes the assigned value. In addition to the data type, information can also be given as to whether the target is an action. All information is then processed by the context. This searches all currently available variables in the program code and returns the most likely reference. The distance to the current position in the code is taken into account as an additional quality criterion if the name is not explicitly mentioned. In these cases, if no existing reference can be assigned, it creates a new field with the data type of the assigned value. The procedure of resolving a target reference is shown in Figure 8.

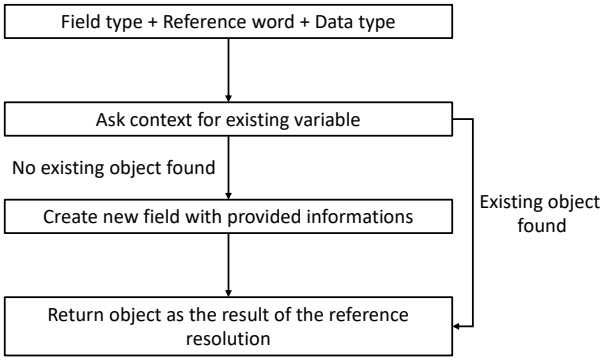

Figure 8. A reference resolution specified as the target of a directed action.

C. Context

The context module is the brain of the system and knows both the program sequence as well as the current position in it and offers functions to search for elements in it. Similar to [18], we solve an ordering problem that arises in natural-language programming. End users provide expressions involving ”before”, ”after”, ”while”, ”at the end”, and others. Our module represents the interface to the current excel document and thus allows the access to cells or area therein. It also knows the statement history and has a semantic understanding of the program code. This means that it knows at which position something was inserted last and what consequences this has on the accessible variables. Therefore, assigning an existing object in the program to a specified reference is an important task of the context. The procedure for resolving field references is shown in Figure 9. This function is called by the respective services as required.

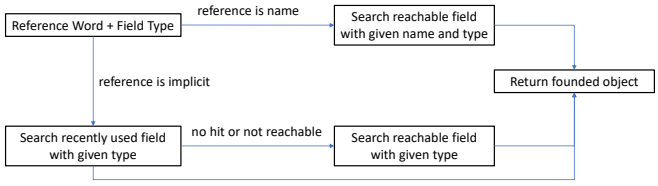

Figure 9. Sequence for resolving a reference in the context.

D. Data structure

The foundation on which both the context and the service structure work is a data structure which represents an object-oriented illustration of a programming language. Program

 ISBN: 978-1-61208-616-3

sequences, actions, control structures and fields are implemented as classes. Functions are implemented as program sequences and contain references to the instructions placed in them. Analogous to this, control structures are constructed, which, depending on the type, additionally have a condition (`branches` or `while-loop`) or a reference to a quantity (`for-loop`). Actions define a processing rule and have appropriate references to fields that are necessary for execution. According to this concept, a program sequence can be represented as an object tree. The advantage of this implementation is the simple management and expansion of a resulting program sequence. Finally, in order to display the program code of the detected algorithm, each of the classes defines a method which dynamically generates the pseudo code of it. The execution is implemented in the same way. Each program component can be executed atomically. As a result of the structure as a tree, the entire program sequence can be drawn or executed by calling the respective method of the root object.

V. EVALUATION

In order to be able to make a statement about the quality of the system, the user study should be considered in addition to the unit tests carried out during the implementation.

A. Unit Tests

Due to the high complexity and complex dependencies, the active ontology was implemented with the process of test-driven development. The identification of individual concepts is ensured by a total of 132 defined entries. This results in a test coverage of 86%. The tests check whether the ontology delivers the expected output to an input. 57 test cases verify easy to recognize instructions, such as definitions, actions and unconditioned loops. 39 tests check the identification of various conditions. 36 defined inputs test the ontology for the detection of more complex instructions such as linking an action to a loop or condition.

In addition to the assurance of the functionality of individual concepts, complete algorithm descriptions are also tested by unit test. The following description of the switching sort algorithm shown in Algorithm 2 is tested.

Algorithm 2 Pseudo code of switching sort.

```
1: procedure SWITCHINGSORT(input)
2:     n ← length(input) − 1
3:     for i ← 0 → n do
4:         for j ← 0 → n do
5:             if input[j] > input[j+1] then
6:                 tmp ← input[j]
7:                 input[j] ← input[j + 1]
8:                 input[j + 1] ← tmp
```

In contrast to the above mentioned tests, the result of the ontology is not checked here, but the algorithm developed in the description has to sort a quantity of positive natural numbers. Thus, the function of the long-term memory and its interaction with ontology is ensured. In summary, the basic functionality is assured with the unit tests and at the same time it is shown that the system is able to process defined natural inputs correctly.

1. "check if any element of the input is higher than the next element"
2. "save it in an auxiliary variable"
3. "then the actual value should be set to the value of the next element"
4. "set the next element to the value of the auxiliary variable"
5. "do this for each element of the input"

B. User Study

In order to gain an impression of the reliability of the system in relation to unknown and unrestricted formulations, a user study with a total of 11 participants was carried out. The participants were undergraduate computer science students and non-native English speakers. With six people, more than half of the participants described themselves as beginners in programming skills. Five participants described their skills at least as advanced. Seven of them have never used our system before. Four of the participants have already used previous version of our prototype in evaluations earlier. We were afraid that the prototype experts could sophisticate our evaluation results. At the end, they struggled the most during the evaluation trying to use our system in the old way, with old natural language structures.

1) Setup: The tasks of the study cover both the recognition of individual concepts as well as the processing of whole algorithm descriptions. In the first part the participants were asked to submit atomic descriptions of the supported programming elements. These includes definitions, assignments, conditions and loops. The second part again contained two tasks. In the first, the users should construct a process to find the largest element in a set of numbers and then write a step-by-step description of this on paper. The second task involved the pseudo code of a switching sort algorithm and claims the realization of this with the system. The reason for this subdivision is the insights that can be gained later. Interesting questionnaires are, for example, what influence the answer and the preview of the algorithm have on the course of the description and the achievement of the target.

2) Results: For a better overview of the results, the supported concepts were divided into three complexity classes. The tasks below are listed and assigned to the respective classes. Instructions describing a class three action are most difficult to detect.

1. Declarations, actions and for-loops:
   - Assigning a cell reference to a variable
   - Remove a value from a quantity
   - Save the value of a reference into a cell reference
   - Iterate over all elements of a quantity
2. Conditions and while-loops:
   - Check whether the value at a given position of a quantity is equal to 5
   - Check whether the value of a variable is already contained in a set
   - Do something until a given quantity is empty
   - Check if the value of a variable is less than three or greater or equal to five

 ISBN: 978-1-61208-616-3

3. Conditioned actions and conditioned loop action:
   - If the value of a variable is smaller than five write it to A3 otherwise store it into B3.
   - Remove all elements less than three from a quantity

Most of the tasks listed are code excerpts which have to be described. Figure 10 shows how reliable descriptions in the respective complexity classes have been correctly recognized. On average, the rate of detection of individual descriptions is 60%. It was noticeable in the analysis of the unrecognized inputs that many of the description could not be recognized due to the use of an unknown synonyms. For this reason, the descriptions were re-evaluated, this time replacing one word with a synonym. This time, a detection rate of 74% was achieved. A comparison of the individual classes is shown in Figure 10. The reason for the, in comparison to the other,

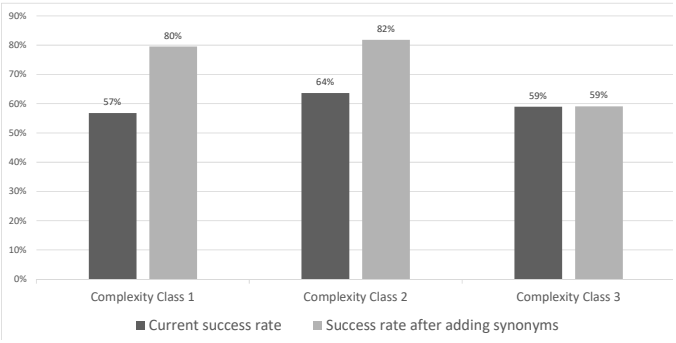

Figure 10. Result of the study for the individual complexity classes with and without synonyms

bad recognition rate in class three was that at the current time side additions are not considered. Therefore, no improvement could be achieved by taking synonyms into account. An input of complicity class three that could not be detected in the evaluation is ”remove all elements from input which are smaller than 3”. The condition is placed in a clause by using the phrase ”which are”. For the ontology, the distance between the referenced field name ”input” and the condition is too large and therefore no relationship is detected. If the condition would be listed without a clause, as in ”remove all elements from input smaller than 3”, the input would have been correctly recognized.

In evaluating the results of a maximum-finding procedure given on paper, only two could be processed and executed correctly. Amazing after this modest success rate was that in implementing a switching sort algorithm with the system, seven out of eleven descriptions could be correctly processed and executed. From this contrast, the advantages of a dialog system and user feedback can be inferred. While this result demonstrate that our system is far from perfect it also shows that it is possible to correctly recognize programming instructions in natural language without restricting them beforehand. Knowing that nearly half of the unsolved tasks stemmed from unknown synonyms which are easy to fix the results we achieved are auspicious.

VI. Related Work

The idea of programming in natural language was first proposed by Sammet in 1966 [19], but enormous difficulties have resulted in disappointingly slow progress. One of the difficulties is that natural language programming requires a domain-aware counterpart that asks for clarification, thereby overcoming the chief disadvantages of natural language, namely ambiguity and imprecision. In recent years, significant advances in natural language techniques have been made, leading, for instance, to IBM’s Watson [20] 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 [21][22]. In 1979, Ballard et al. [23][24][25] 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. [26] has a different orientation. Based on user stories the system tries to derive program structures to support software design. A different approach regarding software design via natural language is taken by RECAA [27]. RECAA can automatically derive UML models from the text and also keep model and specification consistent through an automatic feedback component. A limited domain end-to-end programming is introduced by Le. SmartSynth [28] allows synthesizing smartphone automation scripts from natural language description. However, there is no dialog interaction besides the results output and error messages. One of the last research results have been presented by Wang [29]. They created a convenient natural language interface to perform user tasks. The system uses grammar rules that format natural language into a formal language. However, it is familiar with the pattern matching prototype that we have presented in late 2015 [11].

Paternò [30] introduces the motivations behind end user programming defined by Liberman [4] 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 [8] provides an overview of the research in the area of End-User Programming. As he summarized, many different systems for End User Development have already been realized [31][32][33]. However, there is no system such as our prototype that can be controlled with natural language. During a study in 2006, Ko [31] identifies six learning barriers in End User Programming: design, selection, coordination, use, understanding and information barriers. In 2008, Dorner [34] describes and classifies End User Development approaches taken from the literature, which are suitable approaches for different groups of end users. Implementing the right mixture of these approaches leads to embedded design environments, having a gentle slope of complexity. Such environments enable differently skilled end users to perform system adaptations on their own. Sestoft [35] increases expressiveness and emphasizing execution speed of the functions thus defined by supporting recursive and higher-order functions, and fast execution by a careful choice of data representation and compiler technology. Cunha [36] realizes techniques for model-driven spreadsheet engineering that employs bidirectional transformations to maintain spreadsheet models and synchronized instances. Begel [37] introduces

 ISBN: 978-1-61208-616-3

voice recognition to the software development process. His approach uses program analysis to dictate code in natural language, thereby enabling the creation of a program editor that supports voice-based programming.

NLyze [38], an Add-In for Microsoft Excel that has been developed by Gulwani, Microsoft Research, at the same time as our system. It enables end users to manipulate spreadsheet data by using natural language. It uses a separate domain-specific language for logical interpretation of the user input. Instead of recognizing the tables automatically, it uses canonical tables which should be marked by the end user. Another Gulwani's tool QuickCode [39] deals with the production of the program code in spreadsheets through input-output examples provided by the end user [33]. It automates string processing in spreadsheets using input-output examples and splits the manipulations in spreadsheet by entering examples. The focus of his work is on the synthesizing of programs that consist of text operations. Furthermore, many dialog systems have already been developed. Commercially successful systems, such as Apple's Siri, actually based on active ontology [13], and Google's Voice Search [40][41] cover many domains. Reference resolution makes the systems act natural. However, there is no dialog interaction. The Mercury system [42] designed by the MIT research group is a telephone hotline for automated booking of airline tickets. Mercury guides the user through a mixed initiative dialog towards the selection of a suitable flight based on date, time and preferred airline. Furthermore, Allen [43] describes a system called PLOW developed at Stanford University. As a collaborative task agent PLOW can learn to perform certain tasks, such as extracting specific information from the internet, by demonstration, explanation, and dialog.

VII. Conclusion and future work

In this paper, we presented our work on a natural user interface which enables the end users to program in natural language. Based on active ontologies, programming concepts such as loops, conditionals and statements can be recognized in the analysis of the natural input. A meta model of the recognized concept, which contains relevant information, is then forwarded to the interpreter service provider. Here, the meta model is transformed into the target data structure with the help of a contextual knowledge. This corresponds to the object-oriented representation of a simple programming language. The context finally places the recognized program component in the history and informs the user of the detected action by updating the pseudo code. In this way, an algorithm is generated in an interactive process step for step, which can also be executed at the user's request.

In the evaluation of the prototype it was shown that the system is able to perform the required tasks. However, there is still a lot of work on our system needs to be done. The goal is the independent expansion of the language domain based on the basic vocabulary by means of a dialog system. In the following, such functions, algorithms, control structures and definitions to be recognized are summarized under description. The aim of this work is to ensure that system recognizes the synonyms entered by the user from a known description, learns it, assigns it to the known descriptions and saves it in a user-specific dictionary. This is intended as a local long-term memory for future input. Current work enables end users to describe algorithms and create code sequences as functions. Next step is to enable object-oriented programming [44]. Based on this, end users will also be able to interact with already existing objects, e.g., Excel tables, images, graphs, but also external connections, such as connecting to SQL tables. Such objects should be addressed directly and manipulated by natural-language input. In this case, our system analyzes big data and allows requests from different resources like tables, charts, and databases. End users could ask for information in their natural language that cannot be looked up in one step by the human.

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 [45]. However, programming in natural language remains an open challenge [22]. With natural language, programming would become available to everyone. We believe that it is a reasonable approach for end user software engineering and will therefore overcome the present bottleneck of IT proficients.

References

[1] A. Wachtel, J. Klamroth, and W. F. Tichy, "Natural language user interface for software engineering tasks," Tenth International Conference on Advances in Computer-Human Interactions, March 2017.

[2] W. F. Tichy, "What can software engineers learn from artificial intelligence," Computer;(United States), vol. 20, no. 11, 1987.

[3] ——, "NLH/E: A Natural Language Help System," the 11th International Conference on Software Engineering, 1989.

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

[5] Eurostat, "European statistics database," last accessed March 17, 2018. [Online]. Available: http://ec.europa.eu/eurostat/data/database

[6] Microsoft, "By the numbers," last accessed March 17, 2018. [Online]. Available: https://news.microsoft.com/bythenumbers/planet-office

[7] M. F. Hurst, "The interpretation of tables in texts," 2000.

[8] 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.

[9] B. M. Christopher Scaffidi, Mary Shaw, "Estimating the numbers of end users and end user programmers," in Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, ser. VLHCC '05. IEEE Computer Society, 2005.

[10] R. Abraham, "Header and Unit Inference for Spreadsheets Through Spatial Analyses," in IEEE Symposium on Visual Languages - Human Centric Computing, 2004.

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

[12] A. Wachtel, J. Klamroth, and W. F. Tichy, "A natural language dialog system based on active ontologies," in Proceedings of the Ninth International Conference on Advances in Computer-Human Interactions, 2016.

[13] D. Guzzoni, "Active: A unified platform for building intelligent web interaction assistants," in Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on. IEEE, 2006, pp. 417–420.

[14] A. Wachtel, M. T. Franzen, and W. F., "Context Detection In Spreadsheets Based On Automatically Inferred Table Schema," ICHCI 2016: 18th International Conference on Human- Computer Interaction, World Academy of Science, Engineering and Technology, October 2016.

[15] A. Wachtel, "Programming Spreadsheets in Natural Language: Design of a Natural Language User Interface," Intl Journal on Advances in Software, Vol. 10, no. 3/4. Nice, France., December 2017.

 ISBN: 978-1-61208-616-3

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

[17] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. Association for Computational Linguistics, 2003, pp. 173–180.

[18] T. Hey, "Deriving time lines from texts," in 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'14), ser. ICSE. ACM New York, 2014.

[19] J. E. Sammet, "The use of english as a programming language," Communications of the ACM, vol. 9, no. 3, 1966, pp. 228–230.

[20] 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, 2010, pp. 59–79.

[21] 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.

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

[23] B. W. Ballard and A. W. Biermann, "Programming in natural language:nlc as a prototype," in Proceedings of the 1979 annual conference. ACM, 1979, pp. 228–237.

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

[25] 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, 1983, pp. 71–87.

[26] 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.

[27] S. J. Körner, M. Landhäußer, and W. F. Tichy, "Transferring research into the real world: How to improve re with ai in the automotive industry," in Artificial Intelligence for Requirements Engineering (AIRE), 2014 IEEE 1st International Workshop on. IEEE, 2014, pp. 13–18.

[28] V. Le, S. Gulwani, and Z. Su, "Smartsynth: Synthesizing smartphone automation scripts from natural language," in Proceeding of the 11th annual international conference on Mobile systems, applications, and services. ACM, 2013, pp. 193–206.

[29] S. I. Wang, S. Ginn, P. Liang, and C. D. Manning, "Naturalizing a programming language via interactive learning," arXiv preprint arXiv:1704.06956, 2017.

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

[31] 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.

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

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

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

[35] P. Sestoft and J. Z. Sørensen, "Sheet-defined functions: implementation and initial evaluation," in International Symposium on End User Development. Springer, 2013, pp. 88–103.

[36] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva, "Bidirectional transformation of model-driven spreadsheets," in International Conference on Theory and Practice of Model Transformations. Springer, 2012, pp. 105–120.

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

[38] S. Gulwani and M. Marron, "Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation," in Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014, pp. 803–814.

[39] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," in ACM SIGPLAN Notices, vol. 46, no. 1. ACM, 2011, pp. 317–330.

[40] J. R. Bellegarda, "Spoken language understanding for natural interaction: The siri experience," in Natural Interaction with Robots, Knowbots and Smartphones. Springer, 2014, pp. 3–14.

[41] J. D. Williams, "Spoken dialogue systems: Challenges, and opportunities for research." in ASRU, 2009, p. 25.

[42] S. Seneff, "Response planning and generation in the mercury flight reservation system," Computer Speech & Language, vol. 16, no. 3-4, 2002, pp. 283–312.

[43] J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom, "Plow: A collaborative task learning agent," in AAAI, vol. 7, 2007, pp. 1514–1519.

[44] N. Omar and N. A. Razik, "Determining the basic elements of object oriented programming using natural language processing," in Information Technology, 2008. ITSim 2008. International Symposium on, vol. 3. IEEE, 2008, pp. 1–6.

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

 ISBN: 978-1-61208-616-3