Programming Spreadsheets in Natural Language: Design of a Natural Language User Interface

Alexander Wachtel, Felix Eurich, Walter F. Tichy

Karlsruhe Institute of Technology, Karlsruhe, Germany
Email: alexander.wachtel@kit.edu, felix.eurich@student.kit.edu, walter.tichy@kit.edu

***Abstract*—In this paper, we present the idea to use the natural language as the user interface for programming tasks. Programming languages assist with repetitive tasks that involve the use of conditionals, loops and statements. However, users can easily describe tasks in their natural language. We aim to develop a *Natural Language User Interface* that enables users to describe algorithms, including statements, loops, and conditionals. For this, we extend our current spreadsheet system to support control flows. Although far from perfect, this research might lead to fundamental changes in computer use. 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.**

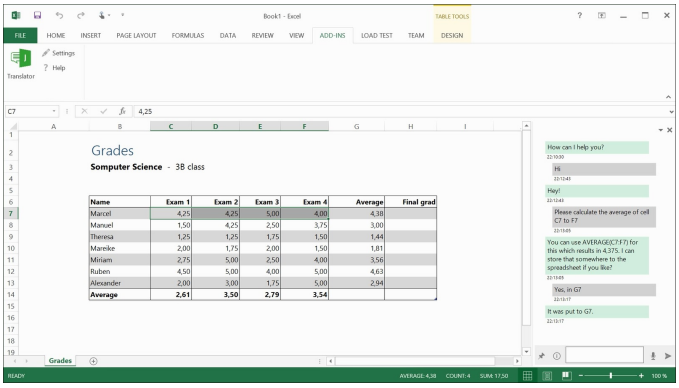
Keywords–Natural Language Processing; End-User Development; Natural Language Interfaces; Human Computer Interaction; Programming In Natural Language; Dialog Systems.

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 it without any knowledge of programming languages. This vision forms the basis for our natural language user interface [1]. Already in 1987, Tichy discussed that 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. Similar to programming languages, our natural language contains all necessary prerequisites to describe an algorithm. However, although you already have the required ability in its core, you have to go through a lengthy learning process in order to master a programming language. Our idea is that, in the future, it will no longer be necessary. The goal is to create a way for the computer to deal directly with natural language. Therefore its necessary to identify and use these similarities between our natural language and the programming language. In general, the system adapts to us and we don't have to go the tedious way through a programming language anymore. Through this paradigm shift 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 [5]. However, programming in natural language remains an open challenge [6].

We are working on the *Natural Language User Interface (NLUI)* that expects natural language input, interprets it in the context of programming and delivers a valid output for this via the dialog system (see Figure 1).



Name	Exam 1	Exam 2	Exam 3	Exam 4	Average	Final grad
Marcel	4,25	4,25	5,00	4,00	4,38	
Manuel	1,50	4,25	2,50	3,75	3,00	
Theresa	1,25	1,25	1,75	1,50	1,44	
Mareike	2,00	1,75	2,00	1,50	1,81	
Miriam	2,75	5,00	2,50	4,00	3,56	
Ruben	4,50	5,00	4,00	5,00	4,63	
Alexander	2,00	3,00	1,75	5,00	2,94	
Average	2,61	3,50	2,79	3,54		

Figure 1. User Interface of the natural language dialog system

In our prototype, we decided to address spreadsheets for several reasons:

- a lot of open data available [7]
- easy to manipulate data
- well-known and well-distributed

In general, spreadsheets have been used for at least 7000 years [8]. Myers [9] and Scaffidi [10] 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 users create hundreds of millions of spreadsheets each year [11].

Our paper is structured as following: Section II presents related work in the research areas of programming in natural language, End-User Programming and natural language dialog systems. Section III describes our work on NLUI: pilot study, dialog system, data representation, analysis of data, and finally, control flows. Section IV evaluates latest prototype in an user study. Finally, Section V presents a conclusion of our topic and future work.

II. Related Work

The idea of programming in natural language was first proposed by Sammet in 1966 [12], 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 [13] computer winning against the two world champions in a quiz game called Jeopardy!, where general knowledge is presented in the form of answers and the candidates must phrase the belonging questions, Apple's Siri routinely answering wide-ranging, spoken queries, and automated translation services such as Google's becoming usable [14][6]. In 1979, Ballard et al. [15][16][17] 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. [18] 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 [19]. 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 [20] allows synthesizing smartphone automation scripts from natural language description. However, there is no dialog interaction besides the results output and error messages.

Initially, Cypher [21] used unstructured text for their research. But later found out that the unstructured approach caused too many false interpretations. Their approach, called sloppy programming, uses a specific grammar, based on an existing set of scripts and allows users to enter something simple and natural. Controlled natural language (CNL) is similar, in that it is simple and natural enough, yet it restricts the grammar, and requires the user to learn the restrictions from examples and by feedback. Gordon [22][23] works on scenario-based programming directly in a controlled natural language. They also believes that a natural language interface to an intuitive programming language may play a major role in programming. [24] translates use-case templates written a CNL to process algebra. Similar to our work, they have implemented the prototype in a Microsoft Word plug-in. It checks adherence of use-case specifications to a CNL grammar and translates them into process algebra. Understanding of natural language by a computer system is a complicated problem and has been studied widely [25]. Command & control systems [26] [27] can retrieve answers that may be given in natural language, or sometimes in more appropriate forms. Smarter applications can take into account the moving speed of the mobile phone and provide a more relevant answer. These Activities like schedule dinner in a restaurant, however, are not programming. Such natural language interactions [28] can be carried out using voice or natural text interfaces. Vadas et al. [29] create runnable programs code from unrestricted natural language. It is inspired by [18] and uses deep semantics derived by a parser for combinatory categorical grammars (CCG). As with NaturalJava, the user must know the target programming language. Furthermore, Mihalcea et al. [30] handles effectively some aspects of procedural programming, e.g., steps and loops.

Paternò [31] 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 [9] 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 [32][33][34]. However, there is no system such as our prototype that can be controlled with natural language. During a study in 2006, Ko [32] identifies six learning barriers in End-User Programming: design, selection, coordination, use, understanding and information barriers. In 2008, Dorner [35] 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 [36] 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 [37] realizes techniques for model-driven spreadsheet engineering that employs bidirectional transformations to maintain spreadsheet models and synchronized instances. Begel [38] introduces 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 [39], 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 [40] deals with the production of the program code in spreadsheets through input-output examples provided by the end-user [34]. 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 [41], and Google's Voice Search [42][43] cover many domains. Reference resolution makes the systems act natural. However, there is no dialog interaction. The Mercury system [44] 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 [45] 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.

III. Natural Language User Interface

In 1979, Ballard et al. [15][16][17] introduced the Natural Language Computer (NLC) that enables end-users to program simple arithmetic calculations using natural language. This section describes different research steps of our work on NLUI prototype.

A. Pilot Study

Initially, we needed to see how real users would interact with spreadsheets in natural language. For this purpose, we recruited subjects and asked them to describe how to solve problems with the aid of a spreadsheet. Spreadsheets with data were provided together with the problems. The subjects were asked to imagine explaining to a human partner how to solve a given problem and to write down what they would say. Even though there were no suggestions to ask the imaginary partner for help, many participants did just that. The problems to be solved were selected from the first four chapters of the textbook Excel 2013 Step by Step [46]. The study consists of 35 questions, six of, which are mathematical problems (i.e., calculation of sum and average, rounding decimal values, copying an existing formula to other cells, conditional functions) and six are data and document manipulation tasks (i.e., inserting and sorting of columns, creating tables and diagrams, labeling cells). Other questions dealt with experience with spreadsheets and familiarity with certain functions.

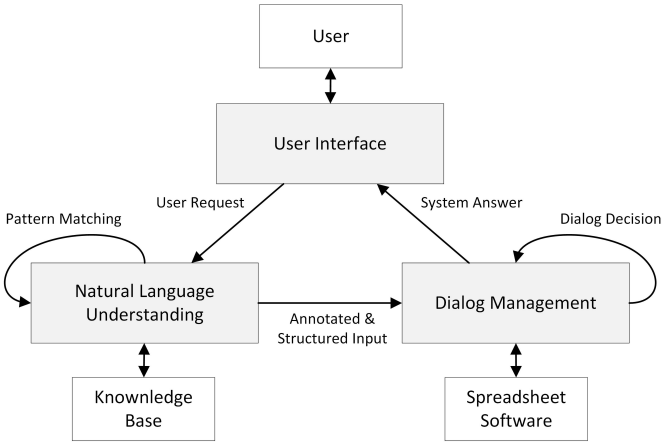


Figure 2. Architecture Overview.

B. Dialog System

Motivated by our pilot study on the natural language prototype, it became clear that a dialog between humans and the computer is required. Suendermann [47] states that in a natural-language dialog system, language perception, and speech production interact with a number of other components. A dialog system is able to communicate with a software system in another language, called meta-language. In general, a dialog system consists of a series of three units, a speech analysis unit for the perception of the natural language input, a dialog management unit for controlling the dialog process, and a speech synthesis unit.

In 2015, first prototype of an assistant has been presented that uses natural language understanding and a dialog management system to allow inexperienced users to manipulate spreadsheets with natural language [48]. The system requests missing information and is able to resolve ambiguities by providing alternatives to choose from. Furthermore, the dialog system must resolve references to previous results, allowing the construction of complex expressions step-by-step. The system architecture consists of a user interface responsible for human interaction, as well as a natural language understanding and a dialog management unit (see Figure 2). In a first step, the natural language understanding unit (NLU) performs essential language analysis relying on a basic vocabulary specifically built to cover the system's domain. Synonyms are substituted using a handcrafted synonym database. Mathematical terms and numerical values as well as references to regions within the spreadsheet are tagged. In the following step, the system groups elements representing a sentence or clause to enable subsequent analysis to perform their work sentence by sentence. With all these adjustments the given example yields a single sentence illustrated in Figure 3 is mapped to a tree structure.

$$\begin{gathered}\text{Please}_{(Request)}\ \text{calculate}_{(Calculation)}\ \text{the}_{(Article)}\\ \text{sum}_{(Operator)}\ \text{of}_{(Possession)}\ \text{column}_{(Positioning)}\\ \text{FinalExam}_{(ReferenceTableColumn)}\ \text{divided}_{(Operator)}\\ \text{by}_{(Specification)}\ 11_{(Number)}\end{gathered}$$

Figure 3. Example of an annotated user input.

Following this, the semantic meaning of any annotated sentence is derived. For this purpose patterns were designed with each pattern consisting of a sequence of keywords and placeholders. The latter can either take a single column, row, cell, number, or arbitrary elements of any type. All patterns are compared with each clause matching the keyword. As part of the given example the pattern *sum* matches the first clause, with keywords *add* and *to*, and placeholders *any* (see Figure 4). The placeholders are filled with the respective elements (*4* and *A1*). By successively matching additional patterns within the placeholder elements, more complex sentences can be transformed into a semantic representation. It builds a tree structure consisting of operators and operands which makes it easy to determine whether a valid arithmetic expression has been provided (see Figure 5). Entirely identified patterns are mapped onto spreadsheet formulas, with the respective placeholders serving as operands, while others have to be processed by the dialog management unit. Incompletely matched patterns are handled by the DMU.

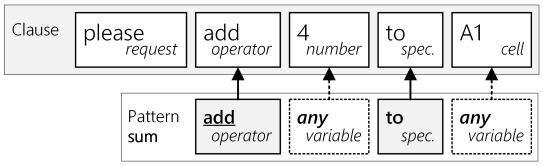


Figure 4. Examplary pattern matching

The purpose of the dialog management unit (DMU) is to deal with the tree structure that has been created by the NLU unit, resolve references, create a valid spreadsheet formula and generate a human-like response to the user input.

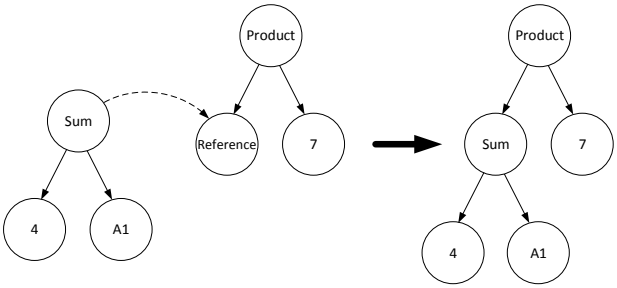


Figure 5. Reference resolution to the previous calculation

Figure 6 shows the process of each user input through three dialog system interpretation stages: contextual, general and fallback. The contextual interpretation runs if the system expects an user answer for missing information or for statement of insert the formula into the spreadsheet. It is always executed before the general interpretation and tries to find a match to the previously posed question within the current user input. For this purpose a set of values from the current conversational context stores all information that might help with dialog decisions. The current context together with any information on previous dialog iterations is presented by a history that keeps track during prolonged conversations with multiple iterations.

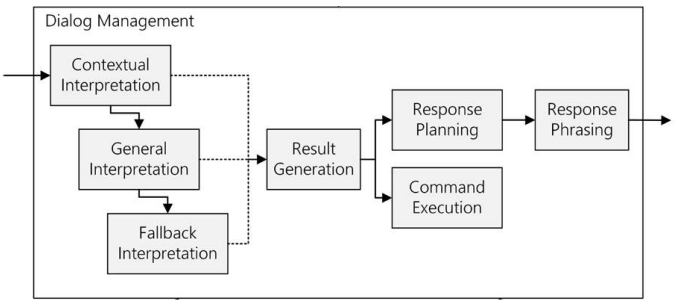


Figure 6. Dialog Management Unit.

Each time a new request is entered by the user, the history is extended by a new context representing the new input. If the user input is not directly related to an earlier iteration, the general interpretation module tries to interpret the entire NLU data structure concerning mathematical operations. If no mathematical topic could be identified, a rudimental fallback interpretation module tries to decide whether the user posed a question or stated a request. Based on the result a universal output is generated that attempts to guide the user towards a well-interpreted input. The main task of the general interpretation is to interpret the roughly structured NLU data structure and to generate a well-formed representation. Since the NLU result might consist of multiple possible ways of understanding the input, this step should be performed independently for each interpretation.

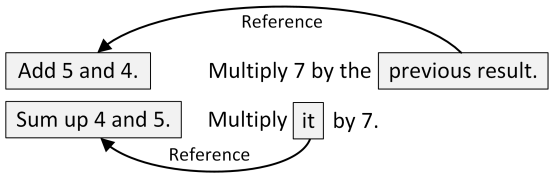


Figure 7. Resolution of references.

Furthermore, the interpretation process deals with resolving references to other clauses inside one sentence or to the topics of previous inputs. In a conversation between humans words like *this*, *that* or *it* usually refer to the most recent subject (see Figure 7). The systems approach works in a similar way and tries to replace a reference by information that fits the conversational context. In order to prevent extensive ambiguities, only elements that turned out to represent a valid formula are taken into account. If multiple results are interpreted as valid, the user has to choose the required calculation.

Besides the interpretations stages, the DMU performs generation of a result and command execution. Furthermore, it responses to the user input as a human-like dialog system. If the operation tree structure found in the current context contains one or more valid operations, the module hands all those over to the response generation module, which will then generate a suitable answer. The response planning module handles the generation of a set of abstract response segments that will be phrased in natural language. In this connection a response phrasing database is used to support the automated phrasing process. A single sentence is represented by a response segment that abstractly specified all required parameters. Those parameters will be later integrated at a certain position in the response. As seen in Figure 8, each segment consists of one or more rules indicated by '*'. A rule consists of one or more alternative phrases that are picked randomly. The same applies for vocabulary entries representing a single word that are indicated by '#'.

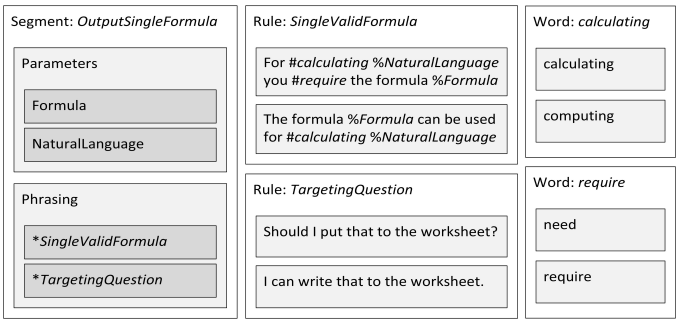


Figure 8. Response Phrasing Database.

Finally, the response phrasing module takes care of generating a string from prepared segments. In case of one valid result without target specification our system choose the segment *OutputSingleFormula*. This segment uses the formula and its natural language expression as parameters and applies the two rules *SingleValidFormula* and *TargetingQuestion*.

The rules choose randomly the vocabulary entries *calculating* and *need*:

For calculating the sum of column FinalGrade divided by 11 you need the formula (SUM(Grades[FinalGrade])/11). Should I put that to the worksheet?

The evaluation of the prototype exceeded expectations. 80% of 170 tasks have been solved successfully. The system helped users to solve tasks and received positive feedback from nearly two thirds of the users. Inspired by the Turing Test [49], the authors asked 17 independent spreadsheet users to formulate requests for particular calculation tasks. Each task was answered by both, the prototype and a human, independently. Afterwards, the participants were encouraged to identify the computer generated response. This however turned out to be surprisingly hard to decide. With 34 decisions made in total, 47.1% falsely identified the dialog system answer as human.

C. Data & Knowledge Representation

In early 2016, the natural language dialog system has been extended with a natural language dialog system based on active ontologies, which enables the user to create and manipulate excel sheets without having to know the complex formula language of excel [50]. Our system is able to resolve references, detect and help resolve ambiguous statements and ask for missing information if necessary. While already quite powerful, this system was not able to handle conditions properly or understand statements involving loops or instructions affecting multiple cells. In this paper, we will present an approach on how to attack these weaknesses.

By adding additional information to an ontology, such as a rule evaluation system and a fact store, it becomes an execution environment instead of just being a representation of knowledge. Sensor nodes register certain events and store them in the fact store. An evaluation mechanism tests the new facts against the existing rules and performs the associated action if one or more rules apply to the stored facts.

In our system, each rule is represented by a separate node in the active ontology. By connecting nodes the developer decides, which type of facts are relevant to which node. In [50], we presented four different types of nodes:

1) Selection-Nodes: These nodes gather all information of their children and pass on the most fitting according to some score, e.g., Instruction node in Figure 9.
2) Gather-Nodes: These nodes gather the information of all children nodes and only create a new fact if all necessary children facts exist, e.g., Binary operation node or Unary operation in Figure 9.
3) Pass-Nodes: These nodes bundle all obtained information of their children into 1 new fact.
4) Sensor-Nodes: These nodes are the ”leaves” of the ontology and react directly to the user input.

Each node-type can be seen as one possible evaluation mechanism. While with these types a developer is able to cover most parts of standard domains of dialog systems one can think of far more complex ones. This is where our new system comes into play. By allowing the developer to use his own evaluation mechanisms, we created an infinite amount of new possibilities what our system is capable of.

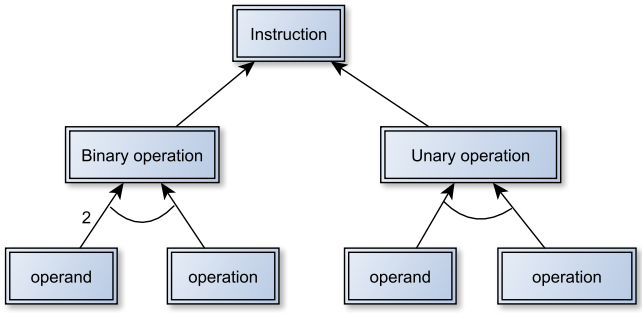


Figure 9. Example of an Active Ontology for mathematical tasks.

Our system consists mainly of two active ontologies: Natural Language Understanding ontology, to interpret the user input, and Natural Language Generation (NLG) ontology, to generate answers (see Figure 10).

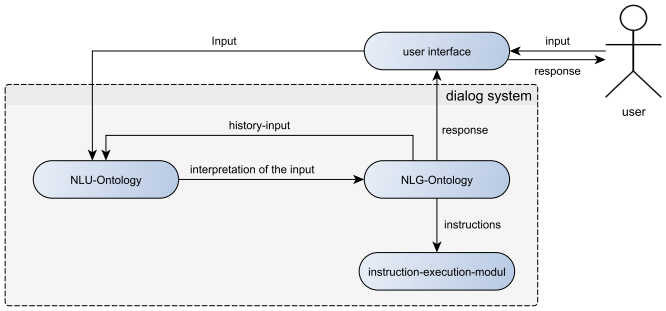


Figure 10. Overview of the active ontology-based dialog system.

The active ontologies recognize several mathematical operations by searching the input, and then recursively building an entire instruction. This instruction is the input to the second active ontology that, based on the type and content of the built instructions, generates a response. When a node gets triggered it builds an answer based on patterns stored in its children. Thus, the answer can be built recursively from reusable predefined patterns. The response generator carries out the process of speech generation in six basic tasks according to Reiter [51]:

- *Content determination*: the basic planning of what should be communicated in the output
- *Discourse planing*: process of imposing the order and structure over the set of messages
- *Sentence aggregation*: grouping of information that should be processed in the response sentence

- *Lexicalization*: choice of words for the sentence
- *Referring expression generation*: insertion of anaphoras to shorten the output
- *Linguistic realization*: bringing the selected words and phrases in a grammatically correct form

Our system roughly follows these steps to generate the answers. Sentence aggregation is almost irrelevant for our use case since the answers of the system are most likely just one or two sentences. The content determination is covered by the use of the input of the NLU ontology. This way the system always knows that information should be communicated in the coming answer. Discourse planning and lexicalization are implicitly stored in the structure and the content of the NLG ontology. The structure of the ontology defines the structure of the generated output and the saved words in the nodes provide this output with the necessary vocabulary. Since the content of the answer is already very specific and very short, referring expressions can be inserted in the sentence structure and do not need to be added in a separate processing step. The structure of the NLG ontology also guarantees a grammatically valid output.

D. Analysis of Data

Interactive Spreadsheet Processing Module (ISPM) [52] 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 (see Figure 11).

	A	B	C	
1	**Table 1: persons**			CAPTION
2	**name**			SUPER HEADER
3	**first name**	**last name**	**age**	HEADER
4	**group A**			GROUP HEADER
5	Sloane	Morgan	37	DATA
6	Dustin	Brewer	33	DATA
7	Valentine	Yates	38	DATA
8	Michael	Gregori	50	DATA
9	**group B**			GROUP HEADER
10	Ina	Hoffman	40	DATA
11	Oliver	Hopkins	27	DATA
12	Damon	Vasquez	22	DATA
13	Mark	Richards	25	DATA

Figure 11. A spreadsheet table annotated with row labels.

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?* (see Figure 12).

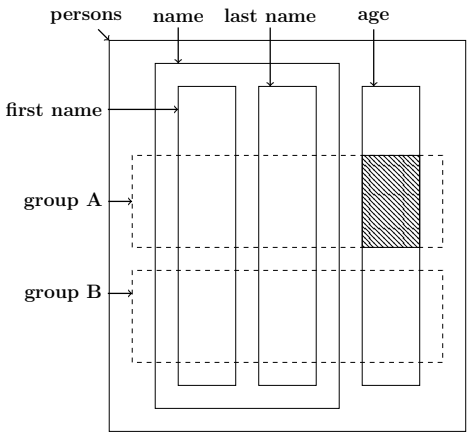


Figure 12. Context detection of user input.

Furthermore, the system enables users to select (88% successfully solved), sort (88%), group (75%) and aggregate (63%) the spreadsheet data by using natural language for end-user software engineering.

E. Action Sets

In 2017, we introduced an interface for natural language dialog system and the Microsoft Excel API [53]. This interface enables users to describe actions in unrestricted natural language interactively and run these actions in the technical domain like Microsoft Excel. We implemented a hybrid system that supports natural language interactions similar as a chat bot and gesture control to follow the users showing actions. For this purpose and also to avoid the lack of inconsistent code documentation, we dynamically export the provided documentation for coding in Visual Studio and provide it in the xml-format to the hybrid system. Users can use already existing actions and provide new action sets to the hybrid system at runtime. For our first prototype of the hybrid system, we concentrate on extending for handling graphs and charts. Evaluated by a user study the hybrid system can successfully resolve more that 80 % of the given user tasks.

Due to the potential complexity and dimensions, the hybrid system extends the consisting natural language approach that manages the interaction through the dialog system and introduces a new graphical user interface providing a better overview and more interaction choices. During runtime, users can switch at any time between these systems. Each interaction increases the associated probability that is used for an assumption about interactions. On each interaction, the dialog system presents the user a limited amount of options, chosen regarding to its probability. To simplify the process for the user only interactions with the last object are available, so no reference is needed. After each interaction the user is asked, if more interactions are welcome, and if so another iteration will be started. If there are no new interaction, the recent set of actions can be saved using an arbitrary name. To avoid duplicate action sets, the module compares the current one with saved sets. If it is duplicate, the user is informed, but still able to save it under new name.

Using the graphical user interface users have access to all options of each object. We defined action sets as arbitrary

sets of API calls learned and executed during runtime (see Figure 13). The system can learn a new action set during runtime from the user via two options: the established dialog system and a graphical user interface. Both systems have the same basic functionality, although the dialog system reduces the options for interactions in each iteration. The dialog system target group are unexperienced users who need more advice from the system and who are quicker using natural language. Advanced users can pick the next interaction freely or users who are quicker by clicking the system are provided a graphical user interface. In case of user input Please, draw a chart, the system tries to detect some implicit steps and asks user for data and chart type to execute this command.

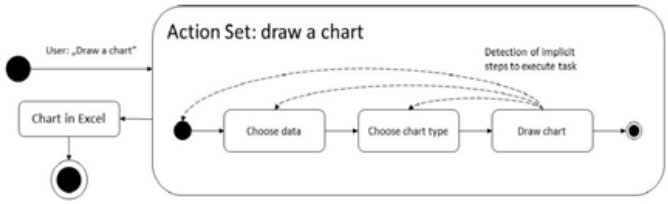


Figure 13. Action set for a simple use input.

F. Control flows

Also in 2017, two new modules have been developed to extend the current system for support of control flows [1]. The first module is capable of handling conditional instructions and the second is able to understand statements that contain loops.

1) Conditional Statements: Conditional instructions are often hard to understand due to their complex grammatical and contextual structure. Also references are complicated to resolve in this kind of sentences. The advantage of conditions is that they have a small set of key-words (such as if, in case of, etc.) that indicate that the user uses a conditional statement. In domain of spreadsheet manipulation to be able to understand, the condition has to result in a boolean operation. These two facts enable us to develop a specialized service dealing with conditional statements. We react to the keywords and try to find a boolean value in the user input. If we can not find any boolean operation, the dialog system asks the user directly for it. After user answers, it is just recognizing, which action the user wants to perform. Unconditional statements were already supported in our older system, so we can rely on it to find the proper action. As already noted, *if* gets recognized as a condition keyword. The system already knows that it is dealing with a conditional statement. *A1 is greater than 3* is a boolean operation and may be used as condition. The trivial statement *save 5 in B1* can be easily recognized as unconditional action and handled by our system (see Figure 14).

2) Loop Statements: Dealing with statements that affect more than one cell can be seen as a looped instruction. In that case, the target of the instruction is the loop variable. Knowing this, we can handle it in a similar way we used for conditional statements. In contrast to conditions, loop do not necessarily have to contain clear keywords. Often times these keywords are hidden within the sentence like *for all*, *for each*, *as long as*. However, there may also be explicit instruction like *do something three times*.

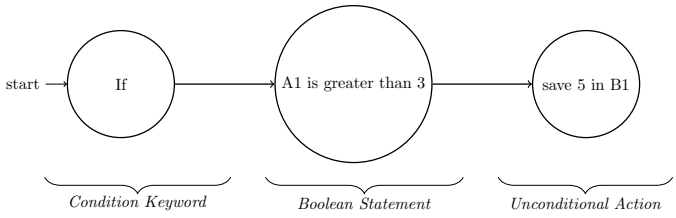


Figure 14. Example for a conditional statement.

Once any of these keywords are recognized, the system has to find the corresponding action and execute it for the given range of cells. In order for our active ontology to be able for recognizing the proper action, we introduced a Looptarget. This is an operator that acts like a normal cell and is able to be recognized by normal actions (such as save, multiply), but at the same time indicates that it can be executed on a range of cells. Once activated by a loop keyword, the loop service reacts to actions containing the regarding targets. In this case, next step is to split input into several instructions for each cell of the given range. This may also be combined with conditional instructions. User input *add 2 to B2 for each cell of column A, which is greater than B2* is correctly handled by our system (see Figure 15).

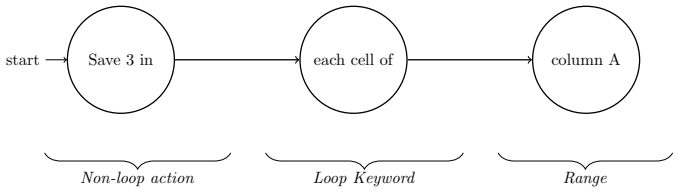


Figure 15. Example for a loop statement.

3) Sample Dialog: To demonstrate the extended features, we present a short dialog between the NLUI and user. For each iteration of the dialog we listed the user input (User), the operation the system executed as consequence of this input (NLU) and the response generated by the system (DMU):

User: Write 30 in all cells of column A

NLU: save(30, A:A)

DMU: Already done. Is there anything else i can do?

User: Subtract 10 from each cell between A1 and A10

NLU: save(A1:A10 - 10, A1:A10)

DMU: Ok, I did that for you.

Can I do something else for you?

User: No

DMU: Alright

. . . After a break, user turns back to the NLUI

User: For each cell of column A which is greater than 25

add 2 to B2

NLU: conditionalAction(A:A>25, save(B2 + 2, B2))

User: If A3 is greater than A2 write 2 in A3 else in B3

NLU: conditionalAction(A3>A2, save(2, A3), save(2, B3))

DMU: Ok, I did that for you.

User: If A5 is greater than B5

NLU: NoAction

DMU: I see you used a condition.

What do you want to do if the condition is true?

User: Save 4 in B3

NLU: conditionalAction(A5 > B5, save(4, B3))

DMU: Everything done as you wished.

IV. Evaluation

In this section, the system will be evaluated in terms of its ability to recognize control structures. The detailed evaluations of previous versions of the system described in III-A and III-B can be found in [48], III-C in [50], III-D in [52], and finally III-E in [53]. To evaluate the quality of control flows solution described in III-F, we conducted an user study. We asked 12 participants to solve given tasks by using our system and recorded their experience with it. In detail, we asked for each task:

- whether they were able to solve the task,
- if the system was helping them to solve the task,
- if the system output was natural,
- and if the system was able to understand their input.

The participants were Non-native English speakers and the majority of them have never used our system before. Most of them stated that they knew and already used excel before, but not on a regular basis.

Since we already evaluated our system for standard arithmetic tasks, response time as well as scalability in our last paper [50], we specifically designed the tasks to test the discussed control flow features, e.g.:

- Insert the specified value 10 into all cells of a column.
- Multiply all cells in a range (between A1 and A10) that are greater than 2 by 3.
- If the value in cell A3 is greater than A2, they should add 2 to B1, else to B2.

The results show that the users where able to solve more than 60% of the tasks at least partially and found our system as useful in over 60% of the cases (see Figure 16). Additionally nearly 70% of the system outputs were considered as natural by the participants. The participants stated in over 50% of the cases that the system did not understand their input. The improvement of the systems output has to be worked on. Overall, the system's quality was rated at 3.33 out of 5 stars, and except for one participant all participants said that they would use our system.

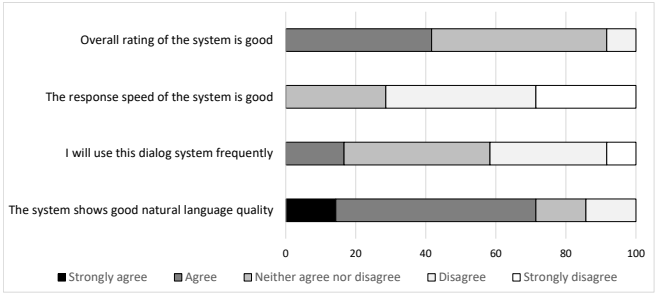


Figure 16. Overall results in %.

While this result demonstrates that our system is far from perfect it also shows that there is added value when using the system especially for inexperienced users. Knowing that nearly half of the unsolved tasks stemmed from the same question and the most common problem were synonym problems, which are easy to fix the results we achieved are auspicious. Since our system will most likely only improve in coming versions due to the growing number of services and the size of our word databases we consider this a promising approach.

V. Conclusion and Future Work

In this paper, we presented the new idea to use the natural language as the user interface. Users can easily describe tasks and delegate it to the system. Often these descriptions contains conditionals, loops and statements (see Section III). To enable the system for end-user development, these parts should be recognized correctly by the system. In the current version of our prototype, the system supports control flows, such as conditionals and loops.

The next goal is to implement valid scripts from natural language input that describes some algorithm like:

- *Find the maximum element of a set:*
  Use an auxiliary variable. Initialize the variable with an arbitrary element of the set. Then, visit all the remaining elements. Whenever an element is larger than the auxiliary variable, store it in the auxiliary variable. In the end, the maximum is in the auxiliary variable.
- *Switching sort of an array:*
  If there are two elements out of order, switch them. Continue doing this until there are no more elements out of order. Out of order means that an element is larger than its right neighbor. The right neighbor of an element x[i] in a vector x is x[i+1].

The challenge is not only the identification of control structures and statements but the recognition of the correct sequence of the given instructions. For considering the relations between user inputs, it is also essential to develop a contextual knowledge. This allows the users to place new statements at any point in the process at runtime and combine several statements into a function, which can be executed at a later time by the system.

In addition to the development of contextual knowledge, the analysis of the input has to be extended by various programming concepts in order to be able to recognize declarations and actions in them. Therefore, the system should (i) enable end-users to give instructions step-by-step, to avoid the complexity in full descriptions and give directly feedback of success (ii) create an abstract meta model for user input during the linguistic analysis and (iii) independently interpret meta model to code sequences that contains loops, conditionals, and statements.

$$Input: A\ sequence\ of\ n\ numbers\ (a_1, ..., a_n)$$

$$\downarrow$$

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.

$$\downarrow$$

Algorithm 1

```
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.
```

$$\downarrow$$

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

The second step after achieving the recognition of algorithms is to allow object-based modeling with natural language. In this case, the system should be able to assign these algorithms as methods to a class. Concepts such as inheritance and relationships between objects should also be identified on the basis of natural language input. With regard to spreadsheets, there are many areas of application that can make the user's life much easier. If you assign value ranges within a spreadsheet to a class, the data can be linked with knowledge from the real world. Through additional assignment of individual methods, questions like "Which cars were built before 2011 and sold more than 5 million times?" can be answered without having to refer to the different areas of the open spreadsheet. The application ranges and extension possibilities of an interface that allows such a mapping between natural language and programming are almost unlimited. However, we are still in the early stages of this development and present the first developments in this direction.

We are also exploring ways to extend the system functionality with the help of the dialog. The system needs to be extended for handling graphs, and charts. Furthermore, there are some properties of tables, which are not considered in the current system and can potentially lead to problems.

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, Vol 20:11, November 1987.

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

[4] H. Liberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm, human-computer interaction series, volume 9," 2006.

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

[6] C. L. Ortiz, "The Road to Natural Conversational Speech Interfaces," IEEE Internet Computing, March 2014.

[7] Eurostat, "European statistics database," 2017, last accessed 2017.11.30. [Online]. Available: http://ec.europa.eu/eurostat/data/database

[8] M. Hurst, "The interpretation of tables in texts," University of Ediburgh, Ph.D., 2000.

[9] B. A. Myers, A. J. Ko, and M. M. Burnett, "Invited Research: Overview End-User Programming," CHI, 2006.

[10] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in Visual Languages and Human-Centric Computing, 2005.

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

[12] J. E. Sammet, "The Use of English as a Programming Language," Communication of the ACM, March 1966.

[13] D. Ferrucci, "Building Watson: An Overview of the DeepQA Project," Association for the Advancement of Artificial Intelligence, 2010.

[14] H. Liu and H. Liebermann, "Toward a programmatic semantics of natural language," Visual Languages and Human Centric Computing, 2004.

[15] B. W. Ballard and A. W. Biermann, "Programming in natural language: NLC as a prototype," Association for Computing Machinery (ACM), Volume 10, 1979.

[16] A. W. Biermann and B. W. Ballard, "Toward Natural Language Computation," American Journal of Computational Linguistics, Volume 6, Number 2, 1980.

[17] A. W. Biermann, B. W. Ballard, and A. H. Sigmon, "An experimental study of natural language programming," International Journal of Man-Machine Studies, 1983.

[18] H. Liu and H. Lieberman, "Metafor: Visualizing stories as code," 10th international conference on Intelligent user interfaces, 2005.

[19] 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," 2014.

[20] V. Le, S. Gulwani, and Z. Su, "SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language," Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys), 2013.

[21] A. Cypher, M. Dontcheva, T. Lau, and J. Nichols, "No Code Required: Giving Users Tools to Transform the Web," Morgan Kaufmann Publishers Inc, 2010.

[22] M. Gordon and D. Harel, "Steps towards Scenario-Based Programming with a Natural Language Interface," Lecture Notes in Computer Science, vol 8415. Springer, 2014.

[23] ——, "Evaluating a Natural Language Interface for Behavioral Programming," IEEE Symp. on Visual Languages and Human-Centric Computing, 2012.

[24] G. Cabral and A. Sampaio, "Formal Specification Generation from Requirement Documents," Notes Theor. Comput. Sci. 195, 2008.

[25] T. Winograd, "Understanding natural language," in Cognitive Psychology 3(1), 1972.

[26] M. A. Hearst, "natural search user interfaces," in Commun. ACM 55(11), 2011.

[27] B. Shneiderman, “Designing the User Interface: Strategies for Effective Human-Computer Interaction,” Addison-Wesley Longman, 1986.

[28] T. Gellar, “Talking to machines,” in Commun. ACM 55(4), 2012.

[29] D. Vadas and J. R. Curran, “Programming with unrestricted natural language,” in Proceedings of the Australasian Language Technology Workshop 2005, 2005.

[30] R. Mihalcea, H. Liu, and H. Lieberman, “NLP (Natural Language Processing) for NLP (Natural Language Programming),” CICLing, LNCS, vol. 3878, 2006.

[31] F. Paternò, “End user development: Survey of an emerging field for empowering people,” in ISRN Software Engineering, vol. 2013, 2013.

[32] 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.

[33] S. Gulwani, W. R. Harris, and R. Singh, “Spreadsheet data manipulation using examples,” in ACM, 2012.

[34] A. Cypher, “Watch what I do: programming by demonstration,” in MIT Press, 1993.

[35] C. Dorner, M. Spahn, and V. Wulf, “End user development: Approaches towards a flexible software design,” in European Conference on Information Systems, 2008.

[36] P. Sestoft and J. Z. Sorensen, “Sheet-defined functions: Implementation and initial evaluation,” 2013.

[37] J. Cunha, J. P. Fernandes, and J. Mendes, “Bidirectional Transformation of Model-Driven Spreadsheets,” Springer Lecture Notes in Computer Science, 2012.

[38] A. Begel, “Spoken Language Support for Software Development,” Ph.D. Thesis, Berkeley, 2005.

[39] S. Gulwani and M. Marron, “NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation,” SIGMOD, 2014.

[40] S. Gulwani, “Automating string processing in spreadsheets using input-output examples,” in ACM SIGPLAN, 2011.

[41] D. Guzzoni, “Active: A unified platform for building intelligent web interaction assistants,” in IEEE, Web Intelligence and Intelligent Agent Technology Workshops, 2006.

[42] J. R. Bellegarda, “Spoken Language Understanding for Natural Interaction: The Siri Experience,” Springer New York, 2014.

[43] J. D. Williams, “Spoken dialogue systems: challenges and opportunities for research,” 2009.

[44] S. Seneff, “Response planning and generation in the MERCURY flight reservation system,” 2002.

[45] J. Allen, N. Chambers, and G. Ferguson, “PLOW: A Collaborative Task Learning Agent,” Association for the Advancement of Artificial Intelligence, 2007.

[46] C. D. Frye, “Microsoft Excel 2013, Step by Step,” O’Reilly Media, 2013.

[47] D. Suendermann, “Advances in Commercial Deployment of Spoken Dialog Systems,” Springer Science Business Media, 2011.

[48] A. Wachtel, S. Weigelt, and W. F. Tichy, “Initial implementation of natural language turn-based dialog system,” International Conference on Intelligent Human Computer Interaction (IHCI), December 2015.

[49] A. M. Turing, “Computing machinery and intelligence,” Mind, vol. 59, 1950, pp. 433–460.

[50] A. Wachtel, J. Klamroth, and W. F. Tichy, “A Natural Language Dialog System Based on Active Ontologies,” The Ninth International Conference on Advances in Computer-Human Interactions, April 2016.

[51] E. Reiter and R. Dale, “Building applied natural language generation systems,” 1997.

[52] A. Wachtel, M. T. Franzen, and W. F. Tichy, “Context Detection In Spreadsheets Based On Automatically Inferred Table Schema,” 18th International Conference on Human- Computer Interaction, rated with Best Paper Award, October 2016.

[53] A. Wachtel, J. Paczia, and W. F. Tichy, “An Interactive Action Set Detection In Natural Language Hybrid System,” Eleventh International Conference on Interfaces and Human Computer Interaction, July 2017.