A Natural Language Dialog System Based on Active Ontologies

Alexander Wachtel, Jonas Klamroth, 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, jonas.klamroth@student.kit.edu, walter.tichy@kit.edu

***Abstract*—Programming today requires years of training. With natural language, programming would become available to everyone and enable end users to program their devices or extend their functionality without any knowledge of programming languages. We present an assistant usable in technical domains that uses natural language understanding, programming step-by-step and an active dialog management system. It allows users to manipulate spreadsheet data by using natural language. We extend our previous system with active ontologies. 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. The system also handles references to previous results and expressions, allowing the construction of complex expressions step-by-step. It also creates new formulas by using End-User Programming concepts and supports the use of repetitive tasks that involve use of conditions and negations. An evaluation shows that the active ontology-based approach resolves 90% of the input tasks which is an increase of 10% over the pattern matching approach.**

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

I. Introduction

Since their invention, digital computers have been programmed using specialized, artificial notations, called programming languages. However, only a tiny fraction of human computer users can actually work with those notations. An alternative would be ordinary, natural language. Ordinary 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 [3]. The idea of programming in natural language was first proposed by Sammet in 1966 [4], 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 [5] 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. However, programming in natural language remains an open challenge [6][7].

Spreadsheets have been used for at least 7000 years [8]. Spreadsheet programs such as Microsoft Excel have become ubiquitous. It is estimated that each year hundreds of millions of spreadsheets are created [9]. In article [1], we presented first prototype of an assistant that uses natural language understanding and a dialog management system to allow inexperienced users to manipulate spreadsheets with natural language. Motivated by a pilot study based on the selected problems from Frey's book *Microsoft Excel 2013* [10] 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.

In this paper, we extended the prototype with an active ontology. The idea of active ontology was first presented by Guzzoni in 2006 [2]. 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. Sensor nodes register certain events and save them in the fact store. An evaluation mechanism tests the new facts against the existing rules and performs the associated actions if one or more rules apply. Our paper is structured as following: section II describes the active ontology framework and the dialog system. Section III evaluates prototype in an user study, followed by performance tests. Section IV discusses both different approaches of pattern matching and active ontologies. Section V presents related work in the research areas of programming in natural language, End User Programming and natural language dialog systems. Finally, section VI presents a conclusion of our topic and future work.

II. Active Ontology-based Dialog System

The prototype is implemented as an add-in for Microsoft Excel and looks like an instant messaging system. The dialog system allows users to express spreadsheet calculations in ordinary typed English and converts the natural language input into arithmetic spreadsheet formulas. All calculations are stored as valid Excel-formulas that allows future manipulation of the spreadsheet data.

A. Active Ontology Framework

We used Guzzoni's approach [2] to develop an active ontology-based framework, further implemented a dialog system relying on this work to understand and interpret ordinary English inputs. The framework generally provides a system that enables the developer to arrange nodes and thus design

 ISBN: 978-1-61208-468-8

an active ontology. Thus nodes can communicate directly with each other. Sensor nodes register based on regular expression evaluations whether they are affected by the given input or not. If activated each node notifies its connected nodes. This sequence recursively repeats itself. On activation, nodes that are defined as "end-nodes" advise the post-processor. This way the post-processor collects all detected events of the active ontology and the user can react accordingly. Guzzoni distinguishes between the following node types:

- Selection node: this node passes the event with the highest reliability score on to its parents, e.g., see *Instruction* node in Figure 1.
- Gather node: this node just sends an event if all children were activated, e.g., *Binary operation* node or *Unary operation*.

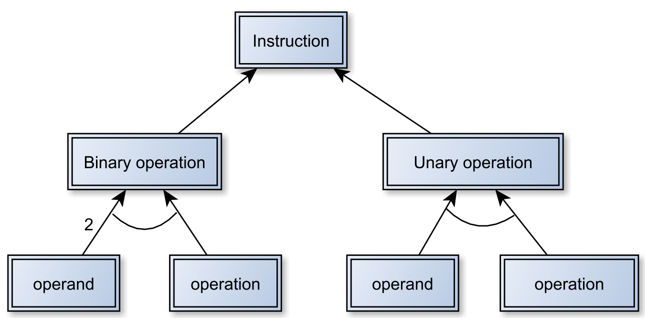

Figure 1. Example of an Active Ontology for mathematical tasks

In addition to Guzzoni, we introduced a new type of node: the generalization node. This node simply collects all incoming events and passes them on to their parent-nodes. It enables the user to combine several events and concentrate them into one single type of event. This is often useful as it enables the user to use different types of events as arguments for one and the same action. In Figure 2, the node operand is a Generalization node. Without this node the sum as well as the difference would have to have three different sum-/difference-nodes for each argument-type one.

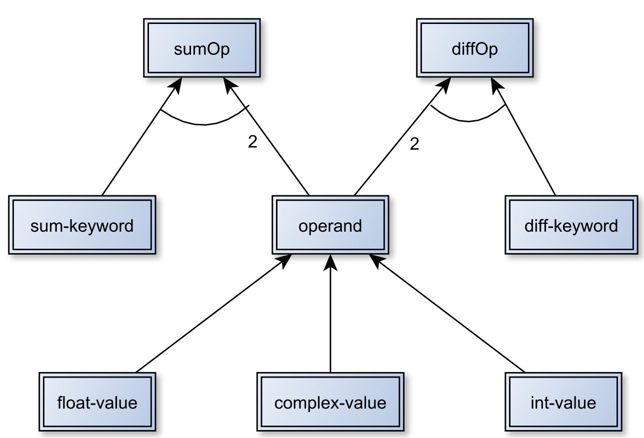

Figure 2. An example of the use of an generalization node

To further clarify how active ontology framework works we discuss the input "save 3 times 2 in A2" as an example. For *save* and *times* the respective sensor node will fire an event signaling that the keyword was found. Additionally two number-values, *3* and *2*, and cell *A2* will be found. Activated by the incoming event the multiplication-node will determine the two most fitting values and thus fire an event signaling a recognized product *(3 * 2)*. A2 is not considered as fitting operand since it is the only valid target for the save operation and thus is reserved for this argument. The product will be recognized as a new operand and thus be send to the save-action-node. Here the node triggered by the keyword *save* will find that an operand *(3 * 2)* as well as a target cell *A2* was found resulting in a new save-action-event. It represents the desired action and can now be performed by the instruction-execution-module leading to the intended outcome.

To ease the act of creating such active ontologies and ensure that there are as few design errors as possible, our framework comes with an editor. The existing active ontology can be displayed either as a graph or a tree structure. This editor not only allows the user to manipulate and create active ontologies but also validates them as far as possible to avoid typical mistakes. Any changes are checked by a validation process for errors. Duplicate names, orphaned nodes, empty regular expressions, missing end node or cycles in the ontology can be detected. Cycles can be especially problematic for recursive algorithms.

B. Dialog System

The dialog system [1] extends the active ontology framework. We use two different active ontologies: Natural Language Understanding (NLU) ontology, to interpret the user input, and Natural Language Generation (NLG) ontology, to generate answers (See Figure 3).

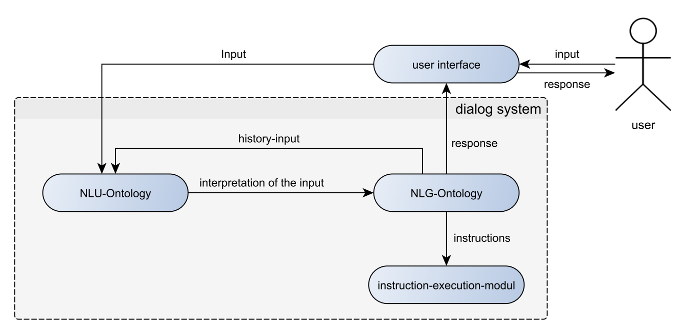

Figure 3. Overview of the active ontology-based dialog system

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 [11]:

- *Content determination*: the basic planning of what should be communicated in the output

 ISBN: 978-1-61208-468-8

- *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.

The system is successful in resolving references to previously provided information, understands negations and asks questions depending on the dialog context:

User: Generate a random number between 1000 and 1500
System: Your input results in the following formula:
randbetween(1000, 1500). Where exactly do you want me to save that in the worksheet?
User: C2
System: Ok, I did that for you
Can I do something else for you?
User: No
System: Alright

User: Calculate C5 minus D5 multiplied with C5
System: Your input may be interpreted in various ways:
a) (C5 - product(D5, C5))
b) product((C5 - D5), C5)
Which one should I use?
User: a
System: Where do you want me to save that in the worksheet?
User: B5
System: Ok, I did that for you.

III. Evaluation

To validate our approach we conducted a user study. Furthermore, several performance tests on the active ontology framework have been done.

A. User Study

Natural language is highly dynamic and expression vary among humans. We asked 16 participants to perform several tasks in Microsoft Excel using solely our dialog system and give feedback on how well the dialog system performed. The tasks covered several complex problems such as resolving references, ambiguities, and conditionals. The study shows that the subjects were able to solve nearly 90% of the tasks and rated the system as helpful in 76% of the tasks. The participants rated the system overall as *good* and *very good* in 15 out of 16 cases. More than 80% of the participants stated they would use the system at least occasionally. Last but not least the answers of the system were appropriate in 88% of the cases according to our participants and 75% of the test persons found the answers to be natural or very natural. User experience with the system is very positive (See Figure 4). 75% of participants said they perceive the dialog system as natural or very natural. The overall quality of our prototype was ranked by 15 of the 16 participants as good or very good.

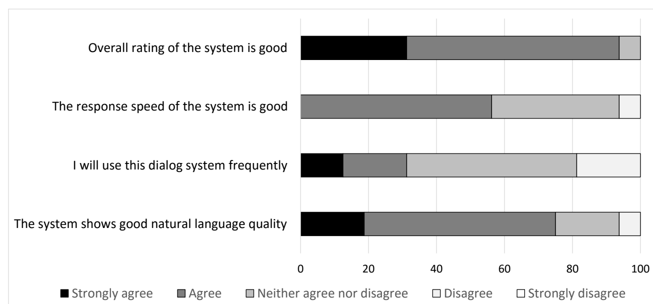

Figure 4. Overall results in %

B. Performance-Tests

One of the most important features of an interactive dialog system is the response time that is required by the dialog system to generate an appropriate answer to a given user input. The user perceives the response time of the dialog system as natural or human-like if it is less than 1s [12]. Larger response times appear as too long and unnatural. In order to evaluate the response time of the dialog system, the response time for every user input during the user study was logged. The average time was 6 ms - the standard deviation was 7.86 ms. Furthermore, our ontology with about 80 nodes and 120 edges needs 140 kB of disk space.

We also designed several automatic tests to explore response times and scaling for large ontologies. In order to avoid the manipulation of the results by choosing specific ontologies we decided to develop a tool that generates random ontologies with certain number of nodes and edges independently. We conducted three different tests measuring the response time. Each test was executed several times on different randomly generated ontologies. The average response time of the active ontology used for our dialog system was less than 6 ms while it took the previous system 150 ms to answer the same questions. It is important to note, that both systems were developed in C# and the tests were run at the same computer with the same operating system (Windows 8.1). Thus we are able to exclude the possibility that the differences are caused by programming language, operating system and/or computation power. In all tests ontologies contained over 10000 nodes and 15000 edges whereby the active ontology framework never exceeded the response time of 600 ms.

1) Length of the input: 10 ontologies all with 100 nodes and 150 edges have been created to test the scaling with length of input. Then, the response time of the system was observed for each ontology for a randomly generated entry. At the same time the length of the input was increased from 100 to 1000 characters. The experiment was repeated 100 times to obtain statistically meaningful values. Figure 5 shows the average values for each input length. While the input size increased tenfold, the response time rose far faster. The values suggest a quadratic growth. These values are negligible for our dialog system, since the input length does not reach the critical time of 1s in almost all cases.

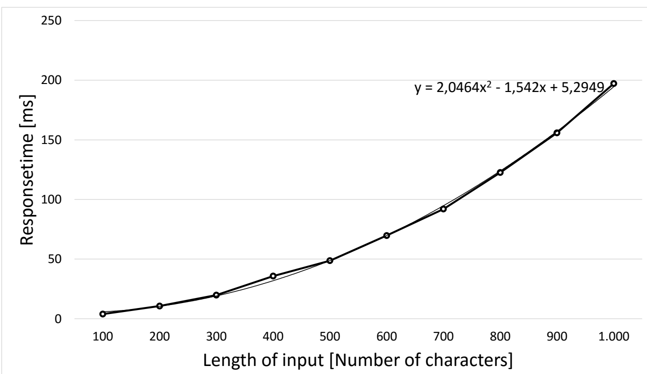

Figure 5. Performance-Test on the length of the input

2) Number of nodes: For this test, 100 ontologies with 100 nodes and 150 edges were created. During the test the number of edges was increased by 10 from 100 to 200. These values were chosen because we determined a ratio of edges to nodes of approximately 1.5 in the ontology. It turns out that in the investigated area, the response time increases linearly with a number of edges and nodes. The slope of the trend line lies below 0.3. This means that the response time increases very slow with increasing interconnectedness (See Figure 6).

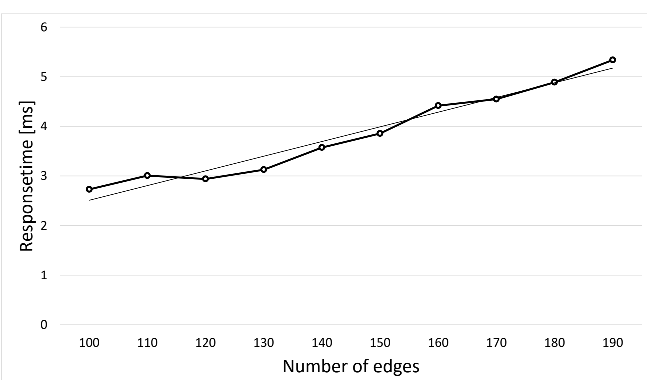

Figure 6. Performance-Test on number of nodes

This demonstrates that our approach is not only capable, but also suited to handle rather large ontologies of nearly every format.

IV. Discussion

Inspired by the Turing Test [13], we 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 asked 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. This result indicates that the prototype is capable of generating suitable responses for sufficiently specific requests within the language domain.

The system [1] is based on matching the user input to patterns that are stored in a database. The iterative dialog system was based on small knowledge base with 92 patterns covering all basic arithmetic operations and assists in the accomplishment of computational tasks. The approach uses a limited vocabulary and a small set of syntactic patterns organized as a grammar. As part of an example, the pattern *sum* matches the first clause, with keywords *add* and *to*, and placeholders *any* (See Figure 7). 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 easier to determine whether a valid arithmetic expression has been provided.

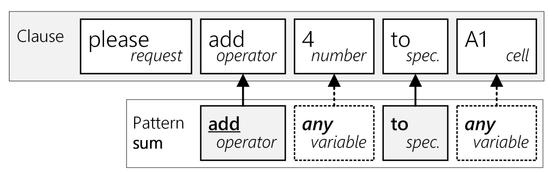

Figure 7. Examplary pattern matching

After the evaluation of an active ontology-based dialog system, we are confident that it has several advantages: it solved 10% more tasks, produces 35% more natural outputs and improves the system by 10% according to the subjects of the user study (See Figure 8).

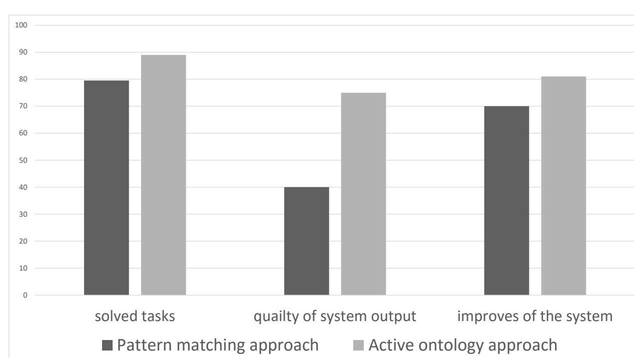

Figure 8. Comparison of pattern matching and active ontology approaches

Furthermore, the active ontology-based system is far more flexible in two ways. First, the design of the application itself is easier and more intuitive when using active ontologies as you can visually arrange the nodes. Second, using patterns the end user is far more restricted in the way talking to the rule-based system in order to be understood. If the user expresses the same action in a slightly different way the provided pattern may not match the input and the system thus fails to understand the user. Active ontologies are much more forgiving in this area as they do not expect the specific order in the input to fit a narrow pattern. This way instructions can be expressed in various ways and still result in the expected behavior.

V. RELATED WORK

Our work combines different research on Programming in Natural Language, End User Programing and Dialog Systems.

A. Programming in natural language

In 1979 Ballard et al. [14][15][16] introduced their Natural Language Computer (NLC) that enables users to program simple arithmetic calculations using natural language. Our prototype extends the idea with a dialog system component for step-by-step construction of complex expression and enables users to perform tasks they otherwise would not be able to accomplish. Although NLC resolves the references as well, there is no dialog system. User feedback has been only provided by the result output. NLyze [17], an Add-In for Microsoft Excel that has been developed at the same time as our prototype, behaves in a similar manner. This approach uses a separate domain-specific language for logical interpretation of the user input. Another approach of programming in natural language is the Pegasus system [18] that creates limited executable programs from natural language using an intermediate knowledge and reference representation. Metafor introduced by Liu et al. [19] has a different orientation. Based on user stories the system tries to derive program structures to support software design. A different approach regarding software design via natural language is taken by RECAA [20]. 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 [21] allows synthesizing smartphone automation scripts from natural language description. However, there is no dialog interaction besides the results output and the output of the error messages.

B. End User Programming

The main question in this area of research is, how to allow users to interact with the computer more easily and allow general users, who are not software developers and have no access to the source code, to program a computer system [22]. In 2009 Myers [23] provides an overview of the research in the area of End-User Programming. Nearly 90 million people in US use computers at work, 50 million use spreadsheets at work, 12 million considered themselves programmers in a self-assessment and only 3 million people are professional programmers. As Myers summarized, many different systems for End-User Programming have already been developed - Whyline, Visual Programming or programming by example and programming by demonstration for spreadsheets. However, there are no End-User Programming systems such as our prototype that can be controlled with natural language. During a study in 2006 Ko [24] identifies six learning barriers in End-User Programming systems: Design, Selection, Coordination, Use, Understanding and Information barriers. Jones [25] encapsulates computation in spreadsheets as a function and proposes a mechanism to define functions using spreadsheet cells, formulas and references. Sestoft [26] extends it by increasing 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. Burnett [27] shows with research language Forms/3 that graphics output, procedural and data abstraction can be supported in the spreadsheet paradigm. Cunha [28] realizes techniques for model-driven spreadsheet engineering that employs bidirectional transformations to maintain spreadsheet models and synchronized instances. Begel [29] 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.

C. Natural Language Dialog Systems

Many dialog systems have already been developed. Commercially successful systems, such as Apple's Siri, actually based on Active Ontology approach [2], and Google's Voice Search [30][31] are characterized by the cover of many domains. The reference resolution makes the systems also acts very natural. However, there is no dialog interaction besides the results output and the output of the error messages. The Mercury system [32] 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 [33] 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.

VI. CONCLUSION AND FUTURE WORK

We present the implementation of an active ontology-based prototype for an assistant that uses natural language understanding and an active dialog management system. It allows inexperienced users to manipulate spreadsheet data by using the natural language. Programming languages provide loops and conditionals. Our prototype already supports conditional expressions. The implemented active ontology framework is robust with large and dense ontologies. The response time of the framework containing 1000 nodes is less than 1 ms. The entire dialog system runs faster than the previous version based on pattern matching - execution time was reduced from 150 ms to 6 ms. Furthermore, the system resolves more tasks correctly and shows better user experience. An evaluation shows that the active ontology-based approach resolves 90% of the input tasks which is an increase of 10% over the pattern matching approach.

However, plenty of work still needs to be done. The goal is to implement Excel scripts called macros from natural language input. We are exploring ways to extend the system functionality with the help of the dialog. Furthermore, the system needs to be extended for handling graphs and charts,

 ISBN: 978-1-61208-468-8

filtering and sorting the spreadsheet data, and supporting loops. Overall, we will implement a module that uses machine learning techniques for context interpretation within spreadsheets and connects natural language to the data in the spreadsheets. That module would enable end users 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?* Perhaps the most important insight is the following: in the past, computers were expected to follow instructions blindly, without a notion of right or wrong or what users expected. With natural language, programming would become available to everyone. We believe that systems like our prototype take first steps in the right direction and are a reasonable approach for end user software engineering, and will help to overcome the present bottleneck of professional developers.

References

[1] Alexander Wachtel, Sebastian Weigelt, and Walter F. Tichy, ”Initial implementation of natural language turn-based dialog system”, International Conference on Intelligent Human Computer Interaction (IHCI), 2015.

[2] Didier Guzzoni, Charles Baur, and Adam Cheyer, ”Active: A unified platform for building intelligent web interaction assistants”, IEEE Web Intelligence and Intelligent Agent Technology Workshops, 2006.

[3] Walter F. Tichy, Mathias Landhäußer, Sven J. Körner. Universal Programmability - How AI Can Help. 2nd International NSF sponsored Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, May 2013.

[4] Jean E. Sammet, ”The Use of English as a Programming Language”, Communication of the ACM, Volume 9 Issue 3, March 1966, pp. 228–230.

[5] David Ferrucci et al., ”Building Watson: An Overview of the DeepQA Project”, Association for the Advancement of Artificial Intelligence, Volume 31 Issue 3, 2010, pp. 59–79.

[6] Hugo Liu and Henry Liebermann. ”Toward a programmatic semantics of natural language”, IEEE Symposium on Visual Languages and Human Centric Computing, September 2004, pp. 281–282.

[7] Charles L. Ortiz, ”The Road to Natural Conversational Speech Interfaces”, IEEE Internet Computing, Volume 18, Number 2, pp. 74-78, Mar.-Apr. 2014.

[8] Matthew F. Hurst, ”The interpretation of tables in texts”, PhD. Thesis, University of Edinburgh, 2000.

[9] Robin Abraham, ”Header and Unit Inference for Spreadsheets Through Spatial Analyses”, IEEE Symposium on Visual Languages and Human Centric Computing, September 2004, pp. 165–172.

[10] Curtis D. Frye, ”Microsoft Excel 2013, Step by Step”, O’Reilly Media Inc., 2013.

[11] Ehud Reiter and Robert Dale, ”Building applied natural language generation systems”, Natural Language Engineering, Volume 3 Issue 01, March 1997, pp 57–87.

[12] Sofia Strombergsson, Anna Hjalmarsson, Jens Edlund, and David House, ”Timing responses to questions in dialogue”, Proceedings of INTERSPEECH 2013, pp. 2584-2588.

[13] Alan M. Turing, ”Computing machinery and intelligence”, 1950, pp. 433-460.

[14] Bruce W. Ballard and Alan W. Biermann, ”Programming in natural language: NLC as a prototype”, Association for Computing Machinery (ACM), Volume 10, 1979, pp. 228–237.

[15] Alan W. Biermann and Bruce W. Ballard, ”Toward Natural Language Computation”, American Journal of Computational Linguistics, Volume 6, Number 2, 1980.

[16] Alan W. Biermann, Bruce W. Ballard, and Anne H. Sigmon, ”An experimental study of natural language programming”, International Journal of Man-Machine Studies, January 1983, pp. 71–87.

[17] Sumit Gulwani and Mark Marron, ”NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation”, Proceedings of the 2014 ACM SIGMOD, 2014, pp. 803–814.

[18] Roman Knöll and Mira Mezini, ”Pegasus - First Steps towards a Naturalistic Programming Language”, 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, 2006, pp. 542–559.

[19] Hugo Liu and Henry Lieberman, ”Metafor: Visualizing stories as code”, Proceedings of the 10th international conference on Intelligent user interfaces, 2005, pp. 305–307.

[20] Sven J . Körner, Mathias Landhäußer, and Walter F. Tichy, ”Transferring Research Into the Real World - How to Improve RE with AI in the Automotive Industry”, First International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), 2014, pp. 13-18.

[21] Vu Le, Sumit Gulwani, and Zhendong Su, ”SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language”, Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys), 2013, pp. 193–206.

[22] Henry Liberman, Fabio Paternò, Markus Klann, and Volker Wulf, ”End-User Development: An Emerging Paradigm”, Springer, Human-Computer Interaction Series, Volume 9, 2006, pp. 1–8.

[23] Brad A. Myers, Andrew J. Ko, and Margaret M. Burnett, ”Invited research overview: end-user programming”, Extended Abstracts on Human Factors in Computing Systems, 2006, pp. 75–80.

[24] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung, ”Six Learning Barriers in End-User Programming Systems”, Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC), 2004, pp. 199–206.

[25] Simon P. Jones, Alan Blackwell, and Margaret Burnett, ”A user-centred approach to functions in excel”, 8th ACM SIGPLAN international conference on Functional programming, 2003, pp. 165–176.

[26] Peter Sestoft and Jens Zeilund, ”Sheet-defined functions: Implementation and initial evaluation”, Springer, Volume 7897 of the series Lecture Notes in Computer Science, 2013, pp. 88-103.

[27] Margaret Burnett, John Atwood, Rebecca W. Djang, James Reichwein, Herkimer Gottfried, and Sherry Yang, ”Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm”, Journal of Functional Programming, Volume 11 Number 2, March 2001, pp. 155–206.

[28] Jácome Cunha, João P. Fernandes, Jorge Mendes, Hugo Pacheco and João Saraiva, ”Bidirectional Transformation of Model-Driven Spreadsheets. Springer Lecture Notes in Computer Science”, Proceedings of the 5th International Conference on Theory and Practice of Model Transformations (ICMT), Springer, 2012, pp. 105–120.

[29] Andrew B. Begel, ”Spoken Language Support for Software Development”, IEEE Symposium on Visual Languages and Human Centric Computing, 2004, pp. 271 - 272.

[30] Jerome R. Bellegarda, ”Spoken Language Understanding for Natural Interaction: The Siri Experience”, Springer New York, Natural Interaction with Robots, Knowbots and Smartphones, 2014, pp. 3–14.

[31] Jason D. Williams, ”Spoken dialogue systems: challenges and opportunities for research”, IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU), December 2009.

[32] Stephanie Seneff, ”Response planning and generation in the MERCURY flight reservation system”, Computer Speech & Language, Volume 16, Issues 34, 2002, pp. 283-312.

[33] James Allen et al., ”PLOW: A Collaborative Task Learning Agent”, Proceedings of the 22nd national conference on Advances in Artificial intelligence (AAAI), Volume 2, 2007, pp. 1514–1519.