Karlsruhe Reports in Informatics 2013,15
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Universal Programmability - How AI Can Help

Walter F. Tichy, Mathias Landhäußer, Sven J. Körner

2013

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Universal Programmability - How AI Can Help

Walter F. Tichy, Mathias Landhäußer, Sven J. Körner
tichy|landhaeusser|sven.koerner@kit.edu

ABSTRACT

Everyone should be able to program. Programming in informal, but precise natural language would enable anyone to program and help eliminate the world-wide software backlog. Highly trained software engineers would still be needed for complex and demanding applications, but not for routine programming tasks.

Programming in natural language is a monumental challenge and will require AI and software researchers to join forces. Early results, however, appear promising. Combining natural language understanding and ontological reasoning helps remove defects from requirements statements, transforms requirements into UML models, and might even enable script-like programming in specific, narrow domains. An important precondition for rapid progress in this area are benchmarks that help compare different approaches and stimulate competition among researchers.

1. INTRODUCTION

Everybody owns one or more programmable devices – mobile phones, digital TVs, PCs, tablets, etc. However, only a tiny fraction of the owners of such devices are actually capable of programming them. This means that the major capability of computers, their programmability, is available to only a tiny proportion of users. The reason is, of course, that programming is difficult – it requires the use of esoteric languages, arcane APIs, and years of training. On the other hand, most people are quiet good at explaining. Parents teach their children in an amazing range of skills, workers train colleagues, friends explain various activities from sports to home repair, and self-help books abound. What if programming was as easy as explaining?

Programming by everyone sounds far-fetched, but by applying artificial intelligence approaches we found that certain types of software artifacts can indeed be produced from natural language input. For instance, software development starts with customer requirements: a skilled analysts elicits and records users' wishes. Once the requirements are complete and correct enough, engineers start transforming the requirements into models and later into executable code. We found that not only can requirements be analyzed automatically for defects such as ambiguity or incompleteness, but requirements can also be transformed automatically into UML models. The reverse transformation is also possible, keeping requirements and models in synchrony. Certain tests can also be generated, by adapting them from similar situations. These software artifacts are only auxiliary to the desired end product, i.e., executable code, but we are now exploring whether simple scripts could be derived from step-by-step instructions formulated in natural language, limited to a narrow domain.

Natural language understanding and processing is a vital part of this vision. If we want to lower the barrier for programming, we need to liberate programming from artificial programming languages and instead rely on people's innate ability of explaining and describing concepts and processes in their mother tongue. With programming in natural language come all the problems of natural language, such as ambiguity and implicit context, and these will have to be handled in some way. However, we are convinced that many of the software engineering steps performed manually today can be carried out, or at least supported, by machines. Essentially, computers should support humans in writing software not only on a syntactic, but also on a semantic level.

2. RECENT ADVANCES IN NATURAL LANGUAGE PROCESSING

IBM's Watson is a program capable of answering questions posed in (typed) natural language [1]. In Feb. 2011, Watson competed in the quiz show Jeopardy!. It beat Brad Rutter, the biggest, all-time money winner on Jeopardy!, and Ken Jennings, who had won 74 games in a row. Except for a few categories, Watson consistently outperformed its human opponents. Watson had access to millions of documents, including Wikipedia, literary works, newswires, WordNet, the Yago knowledge base, and other sources, but not the Internet. Watson uses numerous analysis and search algorithms simultaneously to find the most likely answer. Although Watson does not think nor understand, the system shows the power of a huge database. When programming, humans also rely on a lot of background knowledge, such as algorithms, data structures, software components, domain knowledge, and general knowledge. It seems obvious that broad background knowledge will be needed if software is to be produced in response to requirements or commands in natural language. The software engineering community should learn from Watson's success.

Another example is Apple's Siri (actually developed by a startup founded by SRI International). Siri provides a speech recognition interface and can answer questions or make recommendations. For example, one can ask Siri: "Will I need an umbrella tomorrow?" Siri then checks its calendar to find out where its owner is traveling the next day, checks the weather forecast for that location, and then gives the appropriate answer. Quite a bit of logical inference is involved in answering this query. We think that when making everybody a programmer, it is initially enough to provide a textual interface. Speech input can be added at a later time. The main difficulty is not speech processing, but finding or generating usable software artifacts. Search and logical inferences will be indispensable for this.

If we look at software engineering processes, we see that they rely heavily on natural language. Most requirements documents are written in natural language and people use natural language to explain their needs to the software developers [2]. Humans are

trained in explaining things in natural language and in understanding such instructions: We all go to school where we learn things of increasing complexity via natural language. By leveraging knowledge and explanation, many software engineering tasks should become easier (see Section 4).

For the past 7 years, we have been applying AI technology to software engineering. Combining tools, results, and approaches from SE and AI produced encouraging results which are outlined in the following section.

3. FIRST RESULTS

At first, our research in AI for software engineering concentrated on modeling issues. Gelhausen and Tichy developed a system to extract UML domain models from natural language text [3, 4]. They showed that a systematic extraction process can be carried out, if the text is annotated with so-called semantic roles. Gelhausen and Körner subsequently showed that these annotations can be simplified by using ontologies that contain common world knowledge (such as Cyc [5]) [6]. Further research into domain modeling led to automatically generated checklists for model inspection [7]; these checklists have been applied in software engineering courses.

Körner and Brumm demonstrated that natural language understanding and ontological reasoning can help detect linguistic defects in specifications [8, 9, 10]. Their tool RESI (Requirements Engineering Specification Improver) not only checks for common linguistic defects (such as distortions or deletions), but also recommends possible corrections.

Extending Gelhausen's work on model extraction, Körner and Landhäußer developed a system that semi-automatically annotates a specification with semantic roles [11]. Semi-automatically means that in certain ambiguous cases, the annotation engine asks the user for the correct choice. This development closed the gap between the original specification and the model extraction process [12]. Recent work extended the model extraction process to allow for round trip engineering [13]: Changes to the model – or the specification – can be fed back to its counterpart: Text changes lead to model updates and model modifications are reflected back in the text. This facility allows users to evaluate model changes made by a domain analyst; the domain analyst can better determine the estimated impact of change requests. Sinha et al. follow a similar idea for generating UML class diagrams from use case descriptions but use a restricted natural language [14].

Focusing on the reusability of test scripts, Landhäußer and Genaid [15] developed a recommendation system for acceptance tests. It builds a knowledge base from source code (production code as well as test code) and the respective requirements. For new, unseen requirements the system recommends test steps that are likely to be needed (such as setting up a graphical user interface, loading test data, and so on). Others have applied NLP techniques in software engineering, e.g. in programming by example [16], in inferring contracts from API documentation [17], in detecting flaws in API documentation [18] to name a few.

4. PROGRAM SYNTHESIS

Our current research aims at program synthesis from natural language input. To limit the domain, we use Alice, a framework for building 3D environments, animations, and games. Alice is normally programmed using a Java-like programming language [19]. Alice targets programming novices and is based on real-world objects. It has the advantage that everybody understands the domain.

We analyze the Alice library to build an Alice knowledge base (i.e. ontology); this way our system gets to know the available objects and what they can do. The resulting knowledge base can be harnessed when programming in natural language. So far we have developed a component that analyzes existing Alice objects to build the knowledge base.

Presently, we investigate how potential users of the system would describe Alice animations in natural language, by asking them to describe existing animations. Also, we investigate how to enrich the Alice knowledge base with world knowledge in Cyc [5] and linguistic information in WordNet [20]; also we need to infer a hierarchy for the domain ontology's elements as there is no inheritance in Alice. The envisioned system will then analyze an animation's description and automatically derive an Alice script to build it. To do so, the system does not need to understand the meaning of the description word-by-word, but it must find suitable components in the knowledge base that match the description.

The next milestones are generating a static scene (i.e. the setup for an animation) possibly using other modalities (such as user interaction with a mouse) and generating the script for the actual plot. Future work also includes switching the domain (e.g. to programming humanoid robots).

5. BENCHMARKING NATURAL LANGUAGE REQUIREMENTS PROCESSING

There have been a number of earlier attempts to translate natural language requirements into software artifacts (e.g. [21, 22, 23, 24, 25, 26, 27]). However, these attempts all used different examples to test their systems. The result is that objective comparison is impossible. One simply does not know whether one system is better than another. This situation is clearly undesirable.

We therefore are collecting a benchmark, called nlrpBENCH, short for Natural Language Requirements Processing benchmark [28]. The benchmark holds over 40 requirement texts of varying length and difficulty and their transformations to models, animations, etc. The documents come from various sources and range from computer science exams to real-world industrial specifications; we also include examples and solutions from textbooks. nlrpBENCH is available online (`http://nlrp.ipd.kit.edu/`) and fully open for participation.

Benchmarks have been successfully used in other research communities, for example in robotics, speech processing, and computer architecture. Benchmarks speed up progress because they enable comparison of different approaches. They work because they help discard unsuccessful techniques and spotlight the successful ones. We invite the research community to use and extend nlrpBENCH to develop a common standard for language understanding in a software engineering context.

6. CONCLUSION

The stored-program concept, discovered in the 1940s, was the founding invention for the modern IT industry. It has led to an absolutely astounding growth in both hardware and software. It is common to have dozens to hundreds of applications on a single, programmable device such as a smart phone or a laptop. The number of users of programmable devices is in the billions. However, only a tiny fraction of users is actually able to use the central capability of these devices, namely their programmability. If we

find a way to make programming accessible to anyone, then and only then will computing reach its full potential. Software engineering, artificial intelligence, natural language understanding, knowledge extraction are some of many fields that have to be brought together to achieve the vision of universal programmability.

7. REFERENCES

[1] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Nyberg, J. Prager *et al.*, "Building Watson: An Overview of the DeepQA Project," *AI Magazine*, vol. 31, no. 3, pp. 59–79, 2010. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2303

[2] L. Mich, M. Franch, and P. Inverardi, "Market research for requirements analysis using linguistic tools," *Requir. Eng.*, vol. 9, pp. 40–56, February 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1008821.1008824

[3] T. Gelhausen and W. F. Tichy, "Thematic Role Based Generation of UML Models from Real World Requirements," in *Proc. International Conference on Semantic Computing ICSC 2007*, 2007, pp. 282–289.

[4] T. Gelhausen, "Modellextraktion aus natürlichen sprachen : Eine methode zur systematischen erstellung von domänenmodellen," Ph.D. dissertation, Institut für Programmstrukturen und Datenorganisation, Lehrstuhl Programmiersysteme Prof. Dr. Walter F. Tichy, Fakultät für Informatik, Karlsruher Institut für Technologie (KIT), Jul. 2010. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000019366

[5] Cycorp Inc., "ResearchCyc," last visited 09/04/2012. [Online]. Available: http://research.cyc.com/

[6] S. J. Körner and T. Gelhausen, "Improving Automatic Model Creation using Ontologies," in *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute, Ed., Jul. 2008, pp. 691–696.

[7] T. Gelhausen, M. Landhäußer, and S. J. Körner, *Automatic Checklist Generation for the Assessment of UML Models*, ser. Lecture Notes in Computer Science, M. Chaudron, Ed. Springer, 2009, vol. 5421.

[8] S. J. Körner and T. Brumm, "Improving Natural Language Specifications with Ontologies," in *Proceedings of the Twenty First International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute, Ed., Jul 2009.

[9] ——, "Natural language specification improvement with ontologies," *International Journal of Semantic Computing (IJSC)*, vol. 03, pp. 445–470, 2010.

[10] ——, "Resi - a natural language specification improver," *International Conference on Semantic Computing*, vol. 0, pp. 1–8, 2009.

[11] S. J. Körner and M. Landhäußer, "Semantic enriching of natural language texts with automatic thematic role annotation," in *Proc. of the Natural language processing and information systems, and 15th international conference on Applications of natural language to information systems*, ser. NLDB'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 92–99.

[12] W. F. Tichy, S. J. Körner, and M. Landhäußer, "Creating software models with semantic annotation," in *Proceedings of the third workshop on Exploiting semantic annotations in information retrieval (ESAIR'10)*. New York, NY, USA: ACM, Sep. 2010, pp. 17–18.

[13] M. Landhäußer, S. J. Körner, and W. F. Tichy, "Synchronizing domain models with natural language specifications," in *Proceedings of the Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'2012)*, Jun. 2012.

[14] A. Sinha, M. Kaplan, A. Paradkar, and C. Williams, "Requirements modeling and validation using bi-layer use case descriptions," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Springer Berlin Heidelberg, 2008, vol. 5301, pp. 97–112.

[15] M. Landhäußer and A. Genaid, "Connecting user stories and code for test development," in *Proc. of the Third International Workshop on Recommendation Systems for Software Engineering (RSSE 2012)*, Jun. 2012.

[16] S. Gulvani, "Sythesis from examples: Interaction models and algorithms," *WAMBSE (Workshop on Advances in Model-Based Software Engineering) Special Issue, Infosys Labs Briefings*, vol. 10, no. 2, 2012.

[17] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language api descriptions," in *ICSE*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 815–825.

[18] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/*icomment: bugs or bad comments?*/," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 145–158, Oct. 2007.

[19] M. J. Conway, "Alice: Easy-to-learn 3d scripting for novices," Ph.D. dissertation, Faculty of the School of Engineering and Applied Science, University of Virginia, Dec. 1997.

[20] G. A. Miller, "WordNet: A lexical database for English," *Communications of the ACM*, vol. 38, no. 1, pp. 39–41, 1995.

[21] S. P. Overmyer, B. Lavoie, and O. Rambow, "Conceptual modeling through linguistic analysis using lida," *ICSE 2001*, vol. 0, p. 0401, 2001.

[22] H. M. Harmain and R. J. Gaizauskas, "CM-Builder: An automated NL-based CASE tool," in *ASE*, 2000, pp. 45–54.

[23] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of linguistic techniques for use case analysis," *Requirements Engineering, IEEE International Conference on*, vol. 0, p. 157, 2002.

[24] L. Kof, "An Application of Natural Language Processing to Domain Modelling – Two Case Studies," *Int. Journal on Computer Systems Science Engineering*, vol. 20, pp. 37–52, 2005.

[25] V. Ambriola and V. Gervasi, "Processing natural language requirements," in *ASE '97: Proceedings of the 12th international conference on Automated software engineering (formerly: KBSE)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 36–46.

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

[27] A. Fatwanto, "Translating software requirements from natural language to formal specification," in *Computational Intelligence and Cybernetics (CyberneticsCom), 2012 IEEE International Conference on*, Jul. 2012, pp. 148–152.

[28] W. F. Tichy, M. Landhäußer, and S. J. Körner, "nlrpBench: A Benchmark for Natural Language Requirements Processing," manuscript submitted for publication.