Towards Dialog Systems for Assisted Natural Language Processing in the Design of Embedded Systems

(Invited Paper)

Rolf Drechsler*§ Mathias Soeken*§ Robert Wille*
*Institute of Computer Science, University of Bremen
28359 Bremen, Germany
§Cyber-Physical Systems, DFKI GmbH
28359 Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Combining both, state-of-the art natural language processing algorithms and semantic information offered by a variety of ontologies and databases, efficient methods have been proposed that assist system designers in automatically translating text-based specifications into formal models. But due to ambiguities in natural language, these approaches usually require user interaction. However, efficient and intuitive methods aiding this interaction in system design have hardly been considered. In this paper, ideas for integrating dialog systems into the design flow for embedded systems are proposed. This allows for the creation of a seamless design flow from natural language specifications to formal models.**

I. Introduction

In the field of electronic design automation many achievements can be recorded when it comes to transforming one formal description into another one, e.g. a model on the system level into an RTL description [1] which in turn is translated into a gate level netlist [2]. However, a large amount of time is spent on earlier stages in the design flow, e.g. in requirements engineering or in the process of translating the specification into an initial executable description. As a consequence, electronic design automation is experiencing an increasing effort in the development of algorithms based on natural language processing [3], [4], [5], [6].

From an abstract perspective, these algorithms can be categorized into two fundamental groups namely *fully automatic* and *interactive* algorithms. Algorithms from the first group aim for an automatic approach which acts like a black box and does not require any user interaction. Since natural language is highly ambiguous, fully automatic algorithms usually need to put additional restrictions on the input language. As examples, this can be done by predefining sentence skeletons also known as *boilerplates* [7] or by restricting the whole vocabulary and grammatical expressions using e.g. *controlled English* [8]. As a consequence, being able to perform a fully automatic translation from a natural language specification into a formal model is traded off against the necessity to "learn a new language."

In contrast, interactive algorithms do not aim for a fully automatic translation but instead pursue the idea of not restricting the input language at all. This allows the designer or requirements engineer to write specifications without reassuring her- or himself whether the text adheres to some rules. Interactive algorithms analyze sentences using natural language processing techniques and operate by means of a confidence level. The confidence level is usually high if the sentence is clearly written and leaves barely space for different interpretations. In this case, an automatic translation is performed. However, if the confidence level is low the algorithm tries to narrow down the problem to the actual point of conflict and forms a question which is posed to the designer. The answer to this question is then used to clear out ambiguities and is stored for later occurrences.

II. Dialog Systems

In this paper, we present first implementations of dialog systems that can be integrated in interactive tools based on natural language processing techniques. Two possible directions are considered: (1) An *active* approach in which the user itself initiates a dialog with the computer and (2) a *passive* approach in which the computer provides feedback without explicitly interrupting the user in her or his current work. Both directions are illustrated by means of posible applications.

A. Active Approach

The active approach replicates a "real" dialog with the computer, i.e. the user initiates the dialog by means of a sentence and the tool returns a result. The result either summarizes what has been understood by the algorithm or is a question to gather more information in order to clarify uncertainties.

In order to illustrate a possible application, consider a test case containing a CPU and a thread which is described in natural language by the following two sentences:

A CPU spawns a thread.
The thread sends a message.

This is used as input given e.g. to an interactive tool as proposed in [4] and initiates a dialog with the computer. The algorithm takes these sentences and derives a UML class diagram and a UML sequence diagram by means of natural language processing techniques. The obtained diagrams represent the structure of the implementation and an executable test case, respectively. Initiating the dialog with the first sentence leads to responses such as

Detected class *CPU* without attributes.
Detected class *thread* without attributes.
Detected operation *spawn* for class *CPU*.

Since no uncertainties occurred, the algorithm just summarizes the result. However, in case of uncertainties, the algorithm can actively asks for further information. For example, in the second sentence, the algorithm is not able to determine whether or not a class should be created for the noun "message." The dialog system reports that to the user by responding with

"message" cannot be determined as actor or class?

Then, the designer can add a so-called background sentence in order to clarify the issue. Depending on the dictionary or ontology that is used as the back-end, the algorithm can also explain why the sentence has been misunderstood, e.g.

I know "message" as communication. Do you mean message as in a communication (usually brief) that is written or spoken or signaled; "he sent a three-word message"?

Getting continuous feedback from the tool improves the designer's understanding of the specification. Figuring out the weak points in terms of ambiguities will sometimes lead to a revision of the specification such that a more clear document results. This is also beneficial for other team members, since sentences that are hard to understand by a tool are likely to be misunderstood by other humans as well.

B. Passive Approach

Alternatively, we propose a passive dialog system as it is known from software development and compilers. The input by the designer is a natural language text that can possibly contain several sentences. The tool then acts as a compiler and returns its feedback in terms of warnings and error messages back to the designer. When incorporating this idea into an integrated development environment such as Eclipse, the text can be processed while editing it and messages get annotated e.g. by underlining words. The idea is illustrated in Fig. 1.

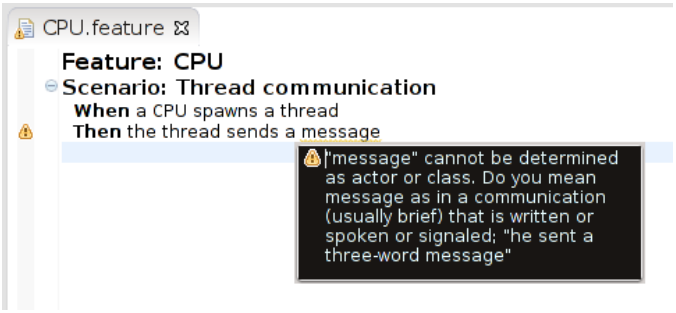

Fig. 1. Passive dialog system

Following this approach, the user does not get interrupted but still is continuously supported with useful information in order to improve the specification. In case all warnings and errors have been resolved, the tool can generate the desired model which can then be used for further processing steps.

III. Conclusions

We propose to challenge the automatized translation of natural language specifications into formal models by means of interactive tools that do not restrict the input language. In order to deal with possible ambiguities, dialog systems are applied that gather additional information of the designer in order to clarify uncertainties. For this purpose, we illustrate two approaches for implementing such dialog systems. One approach is active and resembles a "real" dialog whereas the second approach is passive and gives accompanying feedback while editing the specification. The active approach can be either implemented by means of a text-based chat-like application or can even be modeled as a real dialog using speech recognition and speech synthesis techniques. The passive approach integrates well with modern concepts known from integrated development environments.

References

[1] G. Economakos, P. Oikonomakos, I. Panagopoulos, I. Poulakis, and G. K. Papakonstantinou, "Behavioral synthesis with SystemC," in *Design, Automation and Test in Europe*, Mar. 2001, pp. 21–25.
[2] R. Murgai, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*. Springer, Jul. 1995.
[3] T. Samad and S. W. Director, "Towards a natural language interface for CAD," in *Design Automation Conference*, 1985, pp. 2–8.
[4] M. Soeken, R. Wille, and R. Drechsler, "Assisted Behavior Driven Development Using Natural Language Processing," in *Int'l. Conf. on Objects, Models, Components, Patterns*, May 2012, pp. 269–287.
[5] I. G. Harris, "Extracting design information from natural language specifications," in *Design Automation Conference*, Jun. 2012, pp. 1256–1257.
[6] R. Drechsler, M. Soeken, and R. Wille, "Formal Specification Level: Towards verification-driven design based on natural language processing," in *Forum on Specification & Design Languages*, Sep. 2012, pp. 53–58.
[7] M. E. C. Hull, K. Jackson, and J. Dick, *Requirements Engineering, Second Edition*. Springer, 2005.
[8] N. E. Fuchs and R. Schwitter, "Attempto Controlled English (ACE)," in *Int'l Workshop on Controlled Language Applications*, Mar. 1996.