Just Formal Enough?
Automated Analysis of EARS Requirements

Levi Lúcio[1], Salman Rahman[1], Chih-Hong Cheng[1], and Alistair Mavin[2]

[1] fortiss GmbH, Guerickestrae 25, 80805 München, Germany
{lucio,cheng}@fortiss.org, salman.rahman@tum.de
[2] Rolls-Royce, PO Box 31, Derby, UK
alistair.mavin@rolls-royce.com

Abstract. EARS is a technique used by Rolls-Royce and many other organizations around the world to capture requirements in natural language in a precise manner. In this paper we describe the EARS-CTRL tool for writing and analyzing EARS requirements for controllers. We provide two levels of analysis of requirements written in EARS-CTRL: firstly our editor uses projectional editing as well as typing (based on a glossary of controller terms) to ensure as far as possible well-formedness by construction of the requirements; secondly we have used a controller synthesis tool to check whether a set of EARS-CTRL requirements is realizable as an actual controller. In the positive case, the tool synthesizes and displays the controller as a synchronous dataflow diagram. This information can be used to examine the specified behavior and to iteratively correct, improve or complete a set of EARS-CTRL requirements.

1 Introduction

When writing requirements for software systems in natural language problems such as ambiguity, vagueness, omission and duplication are common [17]. This is due to the the large gap between natural language and the languages in which code is expressed. Natural language requirements describe a wide range of concepts of the real, abstract and imaginary worlds. By contrast, programming languages are used to describe precise sequences of operations inside a machine. Natural language can be partial, ambiguous and subjective, whilst code can typically be none of those things.

EARS (Easy Approach to Requirements Syntax) is an approach created at Rolls-Royce to capture requirements in natural language [17]. EARS is based on practical experience, but has been shown to scale effectively to large sets of requirements in diverse domains [15, 16]. Application of the approach generates requirements in a small number of patterns. EARS has been shown to reduce or even eliminate many problems inherent in natural language requirements [17]. In spite of its industrial success, we are not aware of any published material describing tool support for EARS. The method is primarily aimed at the early stages of system construction, as a means of providing clear guidance to requirements engineers when using natural language to describe system behavior. Automating the writing and analysis of EARS requirements has not been attempted thus far. It is however reasonable to expect that, due to the semi-formal nature of the EARS patterns, automated analysis of EARS specifications can be implemented to improve software development methodologies already in place at Rolls-Royce and elsewhere.

In this paper we will describe our initial work in the direction of automating the analysis of EARS requirements. As domain of application, we have chosen to focus on the construction of controller software. In particular, the EARS requirements for the controller running

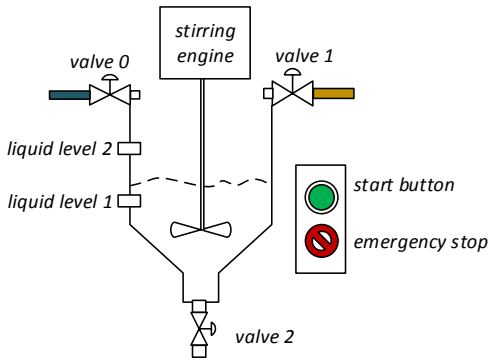

Fig. 1. Liquid Mixing System

example we present in this study have been validated by a requirements engineer at Rolls-Royce. Aside from being industrially relevant, the controller domain lends itself well to analyses and syntheses, given its constrained nature. The contributions described in this paper are as follows:

- An editor for EARS specifications, called EARS-CTRL, based on the projectional editor MPS (Meta Programming System) [2]. Sentences written in our MPS EARS-CTRL editor have the look and feel of pure natural language, but are in fact templates with placeholders for which meaningful terms are proposed to the requirements engineer.
- Automated check of realizability of the requirements as a real controller is provided at the push of a button. Additionally, when the controller is realizable, a synchronous dataflow diagram [14] modelling the specified behavior is generated. This information can be used iteratively to check whether the set of EARS-CTRL requirements correctly express the desired behavior of the natural language requirements written in EARS.

2 Running Example

Our running example for this study is a liquid mixing system. The controller for this system, depicted in fig. 1, is supposed to behave as follows: when the start button is pressed, *valve 0* opens until the container is filled with the first liquid up to the level detected by the *liquid level 1* sensor. *Valve 0* then closes and *valve 1* opens until the container is filled up with the second liquid up to the level detected by the *liquid level 2* sensor. Once both liquids are poured into the container, they are mixed by the *stirring motor* for a duration of 60 seconds. When the mixing process is over, *valve 2* opens for 120 seconds, allowing the mixture to be drained from the container. It is possible to interrupt the process at any point using an *emergency stop* button. Pressing this button closes all valves and stops the *stirring engine*.

3 Expressing and Analyzing Requirements

The first step when writing a set of requirements using EARS-CTRL is to identify the vocabulary to be used. Fig. 2 depicts the glossary for the liquid mixing system we have presented in section 2. The glossary defines the name of the controller being built, the

names of the components of the system that interface with the controller (together with informal descriptions of their purpose), and the sensors and actuators those components make available. Rules expressing relations between signals are also expressed here.

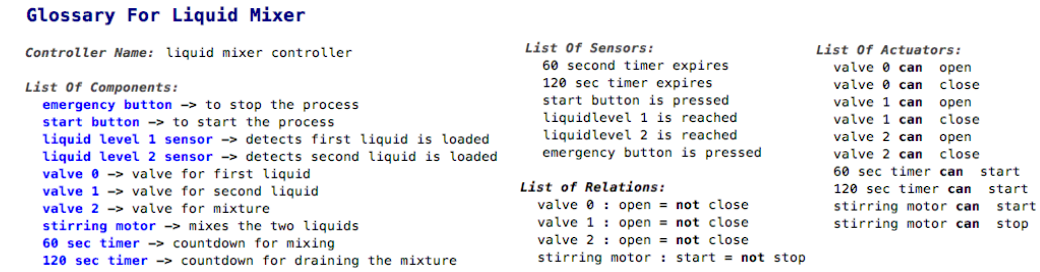

Fig. 2. EARS-CTRL Glossary for the Container Fusing Controller

Once the glossary is defined, the EARS-CTRL requirements can be written. Our editor is built using MPS, a projectional meta-editor for DSL development. The projectional capabilities of the editor make it such that requirements can be edited directly as abstract syntax trees projected onto a textual view. In practice this means that each requirement can be added as an instance of a template with placeholders. These placeholders are then filled by the requirements engineer using the terms defined in the glossary.

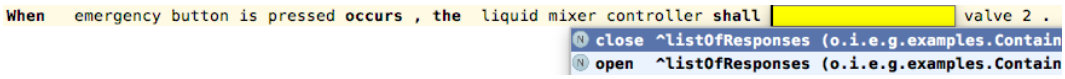

Fig. 3. Example of adding an EARS-CTRL requirement

3.1 Well-Formedness by Construction

In fig. 4 we depict the action of adding an EARS requirement using our editor. Note that two aspects of well-formedness by construction are enforced at this point: firstly, by using EARS templates instances, we guarantee that the form of the requirement is correct; secondly, the editor provides suggestions for the terms that are added to each of the placeholders as a range of possibilities extracted from the glossary. Fig. 3 illustrates some examples for the action associated with the *valve 2* component of the system. Note that in the suggestions associated to this placeholder two constraints are enforced: a) only actions associated with actuators are proposed, and b) the actions for component *valve 2* are limited to the ones that are described in the glossary in fig. 2.

3.2 Realizability Analysis

Well-formedness by construction, as described in section 3.1, guarantees a certain level of correctness of individual requirements. EARS-CTRL provides additional mechanisms for analyzing the interplay of individual requirements in a specification. In particular, at the press of a button the tool can decide whether the set of requirements is realizable as a concrete controller. Note that non-realizability is typically due to conflicting requirements. This analysis is executed by a) transforming EARS-CTRL requirements in LTL (Linear Temporal Logic) formulas, and b) running the GXW synthesis [6] tool autoCode4 [7] via an API to attempt to synthesize a controller for those formulas.

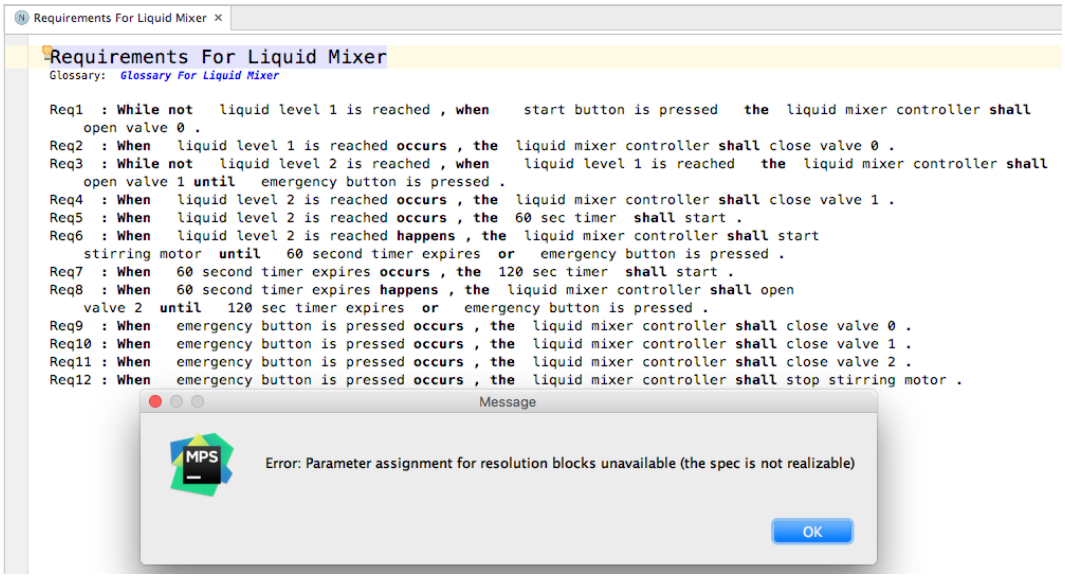

Fig. 4. EARS-CTRL requirements to describe the controller for the liquid mixer system

In fig. 4 we depict a set of requirements[3] for the running example from section 2 that is actually not realizable – as can be understood from the pop-up message in the fig. obtained after running the analysis. When revising the specification, we realized that requirements Req1 and Req9 were in conflict. The reason for this conflict was that, according to Req9, the emergency button can be pressed at any moment thus closing *valve 0*. However, Req1 states that *valve 0* opens when the *start button is pressed*. Thus, logically *valve 0* could be simultaneously open and closed – a contradiction.

```
Req1 : While not liquid level 1 is reached , when start button is pressed the liquid mixer controller shall
    open valve 0 until emergency button is pressed .
```

Fig. 5. Updated Requirement to allow realizing the liquid mixer controller

To eliminate the contradiction we have replaced Req1 in the set of requirements in fig. 4 by the requirement in fig. 5[4] Adding the condition *until emergency button is pressed* to the original version of Req1 disallows *valve 0* being simultaneously open and closed.

When a set of EARS requirements is realizable, EARS-CTRL imports a synchronous dataflow diagram from the autoCode4 tool that describes the behavior of the specified controller. The controller can be visualized inside the EARS-CTRL tool as a block diagram using MPS's graphical rendering capabilities. Due to space limitations, we direct the reader to the project's website [3] for an image of the controller generated for the running example. Note that the synthesized controller is imported into EARS-CTRL as an MPS model, making it possible to further implement automated analyses on this artifact.

[3] For analysability reasons, EARS-CTRL's syntax is slighty different from EARS'. In particular EARS disavows the usage of "until" clauses and composed logical expressions in a requirement.

[4] The requirement in fig. 5 is an instance of template *While A, when B the system shall C until D*. The corresponding LTL is of the form $C \rightarrow (B\ \mathbf{W}\ (D \vee \neg A))$, $\mathbf{W}$ being the weak-until operator.

3.3 The EARS-CTRL Tool

The EARS-CTRL tool is available as a github project [1]. Note that the tool is distributed as an MPS project and requires MPS [2] to be installed as pre-requisite. Together with the functional running example, we distribute with the project the realizable EARS-CTRL requirements for a simple engine controller, a sliding door controller and quiz controller.

4 Related Work

The quest for automatically generating controller implementation from specifications dates back to the ideas of Church [8]. However, it was not until recently that researchers investigated practical approaches to the problem. Methodologies such as bounded synthesis [19] or GR-1 [18], and the combination of compositional approaches [10] have proven to be applicable on moderately-sized examples. Based on these results that stand on solid logical foundations, several projects produced research on the generation of logic formulas from natural language, with the goal of achieving reactive control synthesis from natural language. The ARSENAL project starts from specifications written in arbitrary natural language [11] and also uses GR-1 as the underlying synthesis engine. The work of Kress-Gazit et al. focuses on the synthesis of robot controllers [13]. Their methodology is based on using template-based natural language that matches the GR-1 framework. The work of Yan et al. [20] applies to full LTL specifications and includes features such as guessing the I/O partitioning and using dictionaries to automatically derive relations between predicates (such as $\mathsf{open(door)} = \neg\mathsf{closed(door)}$), in order to detect inconsistencies in specifications.

The workflow presented in this paper, although also targeting the use of natural language, starts with a methodologically different approach. Conceptually, the tool proposes a formal language with a fixed interpretation, while hiding the formality from end-users; in fact an end-user specifies the required system behavior using only natural language. Therefore, for scenarios such as the relation between open(door) and closed(door), the negation relation is not decided during controller synthesis phase but is given during the requirements design phase. Although our tool supports producing generic LTL formulas, our decision for using the autoCode4 tool and the GXW language subset lies on the rationale that, for iterative validation of requirements, it is necessary that designers understand the structure of controllers. For tools [5, 9, 12] supporting GR-1 or bounded synthesis, the synthesized controller is commonly a generated via BDD dumping or via creating explicit state-machines which can have thousands of states, making user interaction and inspection difficult. The work presented here largely draws inspiration from and builds on the knowledge obtained when building the AF3 [4] tool for the model-driven development of software.

5 Conclusions and Future Work

Due to the early nature of this work, two main technical issues remain to be addressed: a) the fact that expressing and analysing complex states such as "the valve is 3/4 closed" or "the quantity of liquid in the container is under quantity X" cannot be reasonably done within EARS-CTRL (due to the boolean representation in autoCode4 of sensors and actuators); and b) lifting the information provided by the analysis engine autoCode4 for debugging EARS-CTRL requirements is currently manually done.

The work described in this paper is an early analysis of the gap between constrained natural language expressed using EARS and logical specifications that can be automatically

transformed into controllers. Note that while the former enables humans to write requirements that are as unambiguous as possible, the latter are developed for computers to process. While these worlds may overlap, they were not necessarily designed to do so.

Ideally, our tool would have as starting point "pure" EARS requirements. However, given the gap mentioned above, we had to slightly adapt "classic" EARS to make it amenable to formal treatment, as briefly mentioned in section 3. The implicit question posed by the title of this paper – whether EARS is *just formal enough* for automated analyses (and syntheses) – is thus partly answered by this work, although additional research is needed. Future efforts will thus concentrate on automatically bridging this gap such that engineers using EARS-CTRL are as unaware as possible of the underlying automatic mechanisms of our tool.

Acknowledgements this work was developed for the "IETS3" research project, funded by the German Federal Ministry of Education and Research under code 01IS15037A/B.

References

1. EARS-CTRL GitHub project. https://github.com/levilucio/EARS-CTRL.git.
2. Meta Programming System. https://www.jetbrains.com/mps/.
3. Wiki for the EARS-CTRL project. https://github.com/levilucio/EARS-CTRL/wiki.
4. V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *ACES-MB (co-located with MoDELS)*, pages 19–26, 2015.
5. A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J.-F. Raskin. Acacia+, a Tool for LTL Synthesis. In *CAV*, pages 652–657. Springer, 2012.
6. C.-H. Cheng, Y. Hamza, and H. Ruess. Structural Synthesis for GXW Specifications. In *CAV*, pages 95–117. Springer, 2016.
7. C.-H. Cheng, E. Lee, and H. Ruess. `autoCode4`: Structural Reactive Synthesis. In TACAS'17, accepted for publication. Tool available at: `http://autocode4.sourceforge.net`.
8. A. Church. Applications of Recursive Arithmetic to the Problem of Circuit Synthesis – Summaries of talks, Institute for Symbolic Logic, Cornell University 1957. *Institute for Defense Analysis, Princeton, New Jersev*, 1960.
9. R. Ehlers. Unbeast: Symbolic Bounded Synthesis. In *TACAS*, pages 272–275. Springer, 2011.
10. E. Filiot, N. Jin, and J.-F. Raskin. Compositional Algorithms for LTL Synthesis. In *ATVA*, pages 112–127. Springer, 2010.
11. S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner. ARSENAL: Automatic Requirements Specification Extraction from Natural Language. In *NFM*, pages 41–46, 2016.
12. B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A Tool for Property Synthesis. In *CAV*, pages 258–262. Springer, 2007.
13. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Translating Structured English to Robot Controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.
14. E. A. Lee and D. G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.
15. A. Mavin and P. Wilkinson. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In *RE*, pages 277–282. IEEE, 2010.
16. A. Mavin, P. Wilkinson, S. Gregory, and E. Uusitalo. Listens Learned (8 Lessons Learned Applying EARS). In *RE*, pages 276–282. IEEE, 2016.
17. A. Mavin, P. Wilkinson, and M. Novak. Easy Approach to Requirements Syntax (EARS). In *RE*, pages 317–322. IEEE, 2009.
18. N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *VMCAI*, pages 364–380. Springer, 2006.
19. S. Schewe and B. Finkbeiner. Bounded Synthesis. In *ATVA*, pages 474–488. Springer, 2007.
20. R. Yan, C. Cheng, and Y. Chai. Formal Consistency Checking Over Specifications in Natural Languages. In *DATE*, pages 1677–1682, 2015.