EARS-CTRL: The Demonstration

Levi Lúcio[1], Salman Rahman[1], Saad Bin Abid[1], and Alistair Mavin[2]

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

Abstract. In this part of the paper, we demonstrate our tool in practice with an example case study of a Quiz Controller. We build controller specification in natural language, automatically generate the controller and then perform validation of the generated controller behavior by utilizing testing and simulation techniques. The demo video can be found at: [URL: youtubevideo]

1 Introduction

In this demonstration, we focus on providing a guidance to the users of our tool. Our EARS-CTRL tool is a github project [1]. An overview of the demonstration steps is as follows, *1*) progressively build a set of requirements for the controller written in english like sentences using the EARS language [4], *2*) automatic synthesizing of the controller by analyzing if the specification is complete (our one click approach) and *3*) simulation and test case generation for performing validation of the controller behavior if it behaves as expected. Performing validation also enables the user to find bugs/errors (e.g., missing requirements) and provides reasoning to the user in order to correct them.

2 The Running Example: Quiz Controller

Our running example for this demo is a Quiz Controller (QC) system controller. The behavior of the sliding door controller is illustrated as a set of EARS-CTRL requirements for the software controller for a QC as shown in figure 1.

```
Requirements for: quiz controller
Glossary:        quiz controller
Temporary path:  solution_root/models

Req1 : While indicator high school is off and indicator professor is off , when pupil button 0 is pressed the
    quiz controller shall blink indicator pupil .
Req2 : While indicator high school is off and indicator professor is off , when pupil button 1 is pressed the
    quiz controller shall blink indicator pupil .
Req3 : While indicator pupil is off and indicator professor is off , when high school button is pressed the
    quiz controller shall blink indicator high school .
Req4 : While indicator pupil is off and indicator high school is off , when professor button 0 is pressed the
    quiz controller shall blink indicator professor .
Req5 : While indicator pupil is off and indicator high school is off , when professor button 1 is pressed the
    quiz controller shall blink indicator professor .
Req6 : When reset button is pressed occurs , the quiz controller shall
    turn off indicator pupil and turn off indicator high school and turn off indicator professor .
```

Fig. 1. Requirements written in *EARS-CTRL* for a Quiz Controller

3 Tool Demonstration

In this section, we discuss the main steps of the process that need to be followed for synthesizing the EARS-CTRL specification into a controller and validating if the controller behaves as expected.

3.1 Glossary building and terms definition

The first step towards writing the controller specifications in natural language using EARS-CTRL is to define the glossary terms. The user is provided with an editor (figure 2) to define glossary terms for, 1) components that interface with the controller, 2) sensors and actuators those components make available to the controller and 3) rules between signals definitions as aliases and invariants.

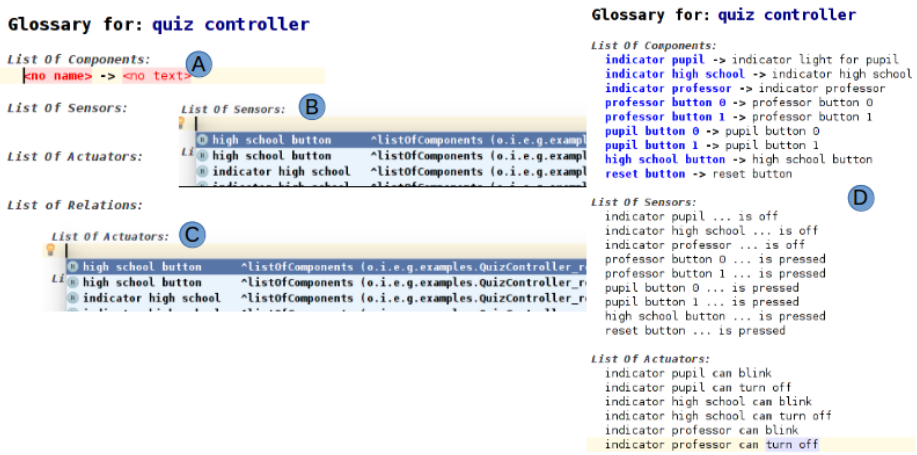

Fig. 2. Step-by-step glossary building for a Quiz Controller: (A) components definition, (B) sensors definition, (C) actuators definition and (D) completed QC glossary

3.2 Building EARS-CTRL requirements for the Quiz Controller

The user is provided with an editor built in MPS [3] to specify the controller behavior. The user can input the requirements simply by filling in the instance of the EARS template with placeholders in the editor. Figure 3 provides step-by-step guidance to write the EARS requirements. In the projectional editor the user is provided with the option (i.e., using *CTRL+Space* keys of the keyboard) to instantiate the EARS-based template with placeholders (*Step A*). After obtaining an instance of the EARS template, the user starts filling in the placeholders with the information (i.e., glossary definitions) (*Step B*). *Step C* of the figure 3 depicts a complete Quiz Controller specification.

3.3 Synthesizing EARS-CTRL requirements

Once the requirements for controller is completely specified, the user can attempt to synthesize the controller. For that, the user can apply the *Transform* intention

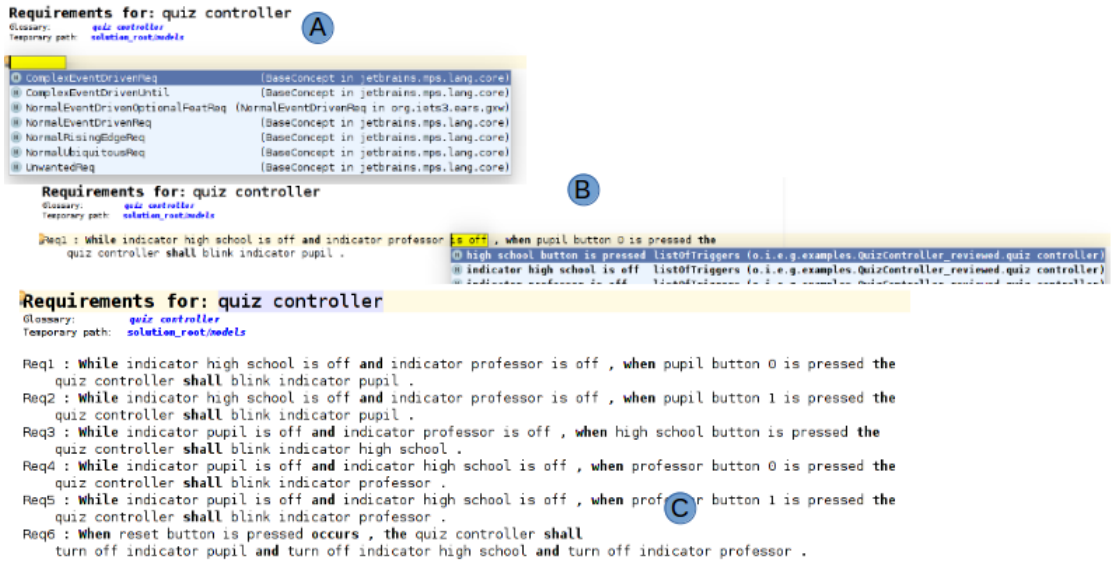

Fig. 3. Step-by-step EARS-CTRL for controller requirements: (*A*) Empty instance of EARS template with placeholders, (*B*) Filling instance of an EARS template with information and (*C*) Completed EARS Specification

(i.e., by using the *Alt+Enter* keys) on the root of the specification (as shown in *part A* of figure 4). The generated outputs after applying the intention comprised of, 1) *Controller Holder*: Data Flow Diagram with blocks and wires (*part B* of figure 4) and 2) *Gate Descriptors*: Pseudo code representing the behavior of the blocks (*part C* of figure 4).

3.4 Simulation and Test Cases for Validation of Controller Behavior

Validation of the generated controller to check if it behaves as expected can be performed by the following techniques, 1) simulating the behavior with Simulink engine [2] as a back-end and 2) automatically generating the test cases. The resulted traces are analyzed in order to check if the controller behaves as expected.

Simulation In order to perform simulation of the generated controller, the user applies the *GenerateSimulinkModel* intention on the root of the *Controller Holder* (i.e., *Step A* in figure 5). After getting the Simulink model generated, the user can apply the intention *AddOutputChecker* (i.e., *Step B and C* in figure 5) to start simulating the controller behavior by providing manual input. For simulation purpose, the user gets the EARS-CTRL panel that allows the user to simulate the controller by providing a sequence of inputs manually. Outputs are incrementally added to the panel as new inputs are provided by the requirements engineer (i.e., *Step D* in figure 5). A Reset Results button enables the user to reset the controller to its initial state.

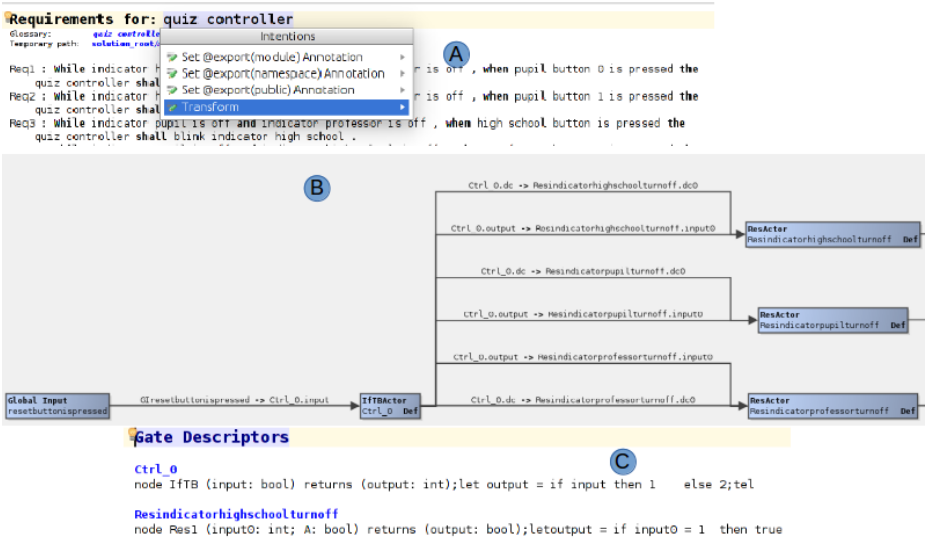

Fig. 4. Controller generation steps: (*A*) applying intention *Alt+Enter*, (*B*) synchronized Data Flow Diagram (DFD) of the controller (an excerpt) and (*C*) pseudo code representing the behavior of the blocks (an excerpt)

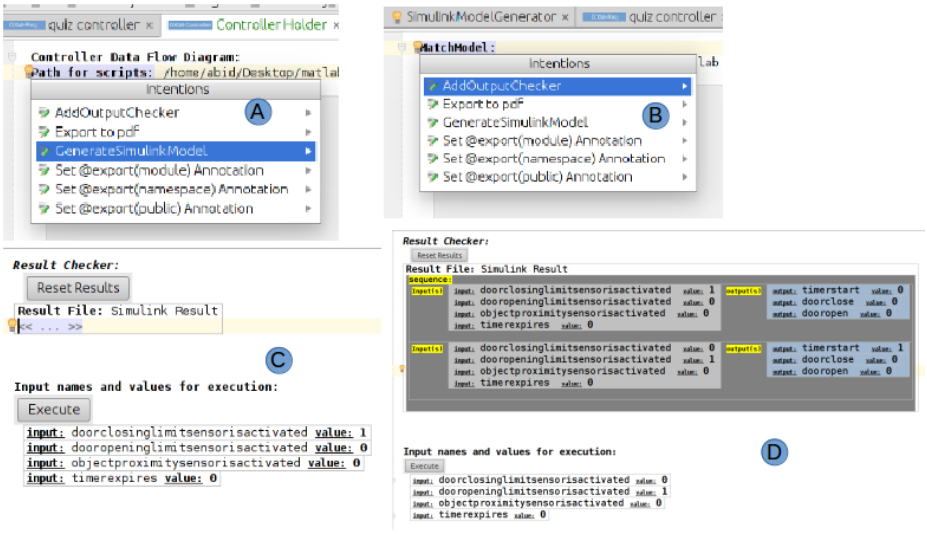

Fig. 5. Simulation steps for validation: (*A*) applying GenerateSimulinkModel Intention, (*B*) applying intention AddOutputChecker , (*C*) generated empty EARS-CTRL panel and (*D*) manually simulating the controller

Automatic Test Cases Generation The user can automatically generate test cases (figure **??**) by setting in the testing parameters. For the test case genera-

tion, the user inputs the parameters in the EARS-CTRL panel. The parameters required for test case generation are as follows, 1) Test Sequence Length(integer value), 2) Allow parallel inputs(true/false) and 3) Allow repeated inputs (true/false).

References

1. Ears-ctrl github project. `https://github.com/saadbinabid1/EARS-CTRLReqAnalysis/`.
2. Mathworks tool suite. `https://de.mathworks.com/`.
3. Meta programming system. `https://www.jetbrains.com/mps/`.
4. A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy approach to requirements syntax (ears). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322, Aug 2009.