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 verification of the generated controller 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 guidence 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 EARS lanaguage [3], *2*) Automatic synthesizing of the controller by analyzing if the specification is complete (our one click approach) and *3*) Simulation and test cases for performing conformance analysis between the EARS-based specification and the generated controller. Performing analysis 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

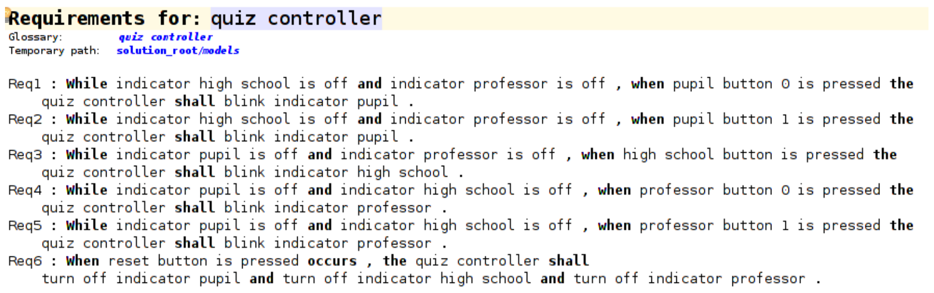
Our running example for this demo is a Quiz Controller (QC) system of PLC based controller. The behavior of the sliding door controller is illustrated as a set of EARS-CTRL requirements for the software controller for a QC as shown in figure 1.

3 Tool Demonstration

In this Section, we discuss the main steps of process that needs to be followed for synthezing and performing the conformance analysis.

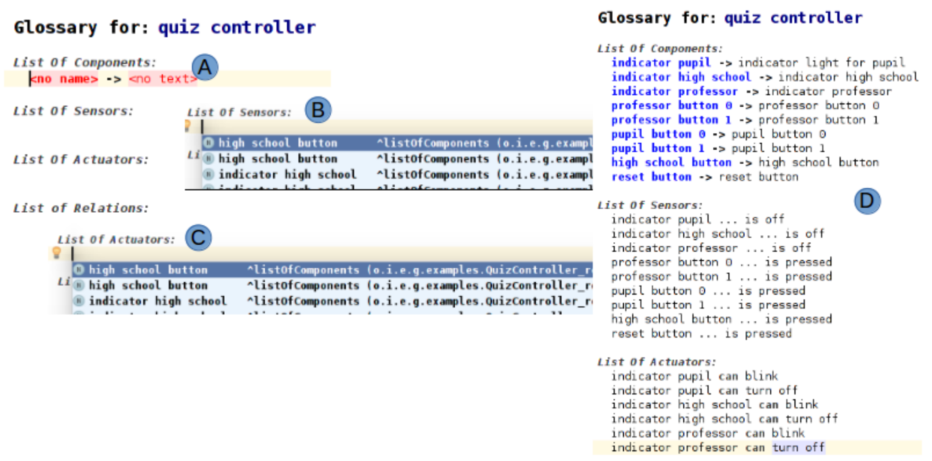
3.1 Glossary building and terms definition

The first step towards writing the controller specifications as a natural language in EARS-CTRL is to define the glossary terms. The user is provided with an editor (figure 2) to define glossary terms for the following, 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.



```
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. *EARS-CTRL* Requiremens for a Quiz Controller



```
Glossary for: quiz controller

List Of Components:
  <no name> -> <no text>

List Of Sensors:

List Of Actuators:

List of Relations:
```

```
List Of Sensors:
  high school button      ^listOfComponents (o.i.e.g.exampl
  high school button      ^listOfComponents (o.i.e.g.exampl
  indicator high school   ^listOfComponents (o.i.e.g.exampl
```

```
List Of Actuators:
  high school button      ^listOfComponents (o.i.e.g.examples.QuizController_r
  high school button      ^listOfComponents (o.i.e.g.examples.QuizController_r
  indicator high school   ^listOfComponents (o.i.e.g.examples.QuizController_r
```

```
Glossary for: quiz controller

List Of Components:
  indicator pupil -> indicator light for pupil
  indicator high school -> indicator high school
  indicator professor -> indicator professor
  professor button 0 -> professor button 0
  professor button 1 -> professor button 1
  pupil button 0 -> pupil button 0
  pupil button 1 -> pupil button 1
  high school button -> high school button
  reset button -> reset button

List Of Sensors:
  indicator pupil ... is off
  indicator high school ... is off
  indicator professor ... is off
  professor button 0 ... is pressed
  professor button 1 ... is pressed
  pupil button 0 ... is pressed
  pupil button 1 ... is pressed
  high school button ... is pressed
  reset button ... is pressed

List Of Actuators:
  indicator pupil can blink
  indicator pupil can turn off
  indicator high school can blink
  indicator high school can turn off
  indicator professor can blink
  indicator professor can turn off
```

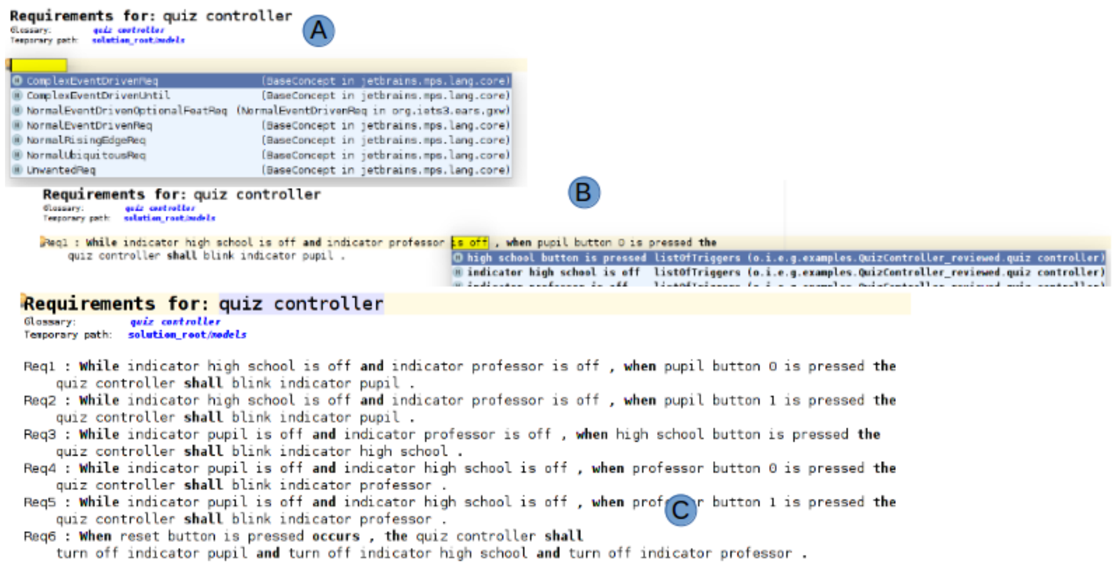
Fig. 2. Step-by-step glossary building for a Quiz Controller: (A) Components definition, (B) Sensors definition, (C) Actuators definition and (D) Completed QC Glossary

3.2 Building EARS-CTRL requirements for the Quiz Controller

The user is provided with an editor built in MPS 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 on 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

The user of the tool can attempt to synthesize the controller once the controller is completely specified. In order to synthesize the controller, the user can apply the *Transform* intention by using the *Alt+Enter* keys on the root of the specification



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

ComplexEventDrivenReq                    (BaseConcept in jetbrains.mps.lang.core)
ComplexEventDrivenUntil                  (BaseConcept in jetbrains.mps.lang.core)
NormalEventDrivenOptionalFeatReq  (NormalEventDrivenReq in org.iets3.ears.gxw)
NormalEventDrivenReq                     (BaseConcept in jetbrains.mps.lang.core)
NormalRisingEdgeReq                      (BaseConcept in jetbrains.mps.lang.core)
NormalUbiquitousReq                      (BaseConcept in jetbrains.mps.lang.core)
UnwantedReq                              (BaseConcept in jetbrains.mps.lang.core)
```

```
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 .
        high school button is pressed  listOfTriggers (o.i.e.g.examples.QuizController_reviewed.quiz controller)
        indicator high school is off   listOfTriggers (o.i.e.g.examples.QuizController_reviewed.quiz controller)
```

```
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 .
```

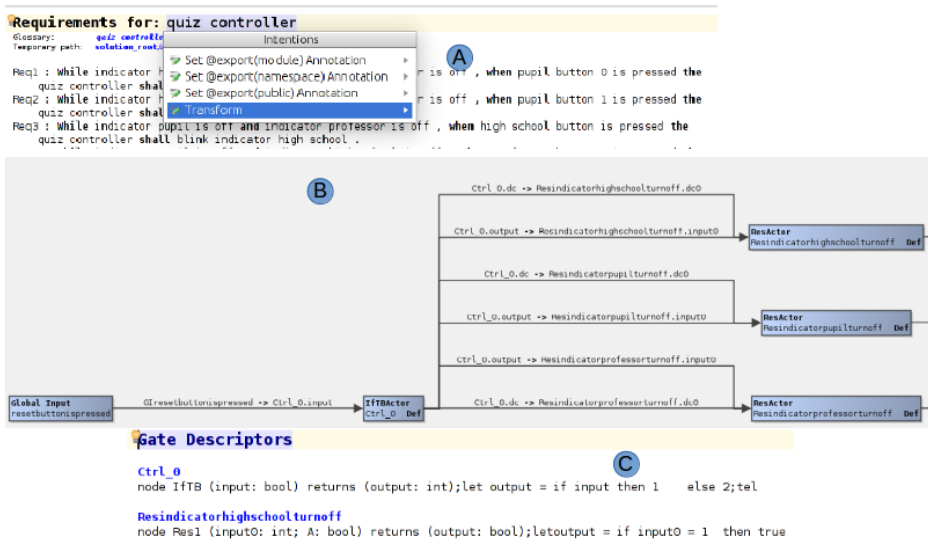
Fig. 3. Step-by-step EARS-CTRL for controller requirements: (A) Empty instance of EARS template with placeholders, (B) Filling instance of an EARS template with information and (C) Completed EARS Specification

(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 (*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 Conformance Analysis

Conformance analysis for verification of the generated controller with respect to its specification 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. The analysis can be performed as follows,

Simulation In order to perform simulation of the generated controller, the user needs to provide the path in the *Controller Holder* model to the folder containing the Simulink blocks (i.e., .m files). More information can be found at the Github project page [1]. The user than 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., Steps 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.

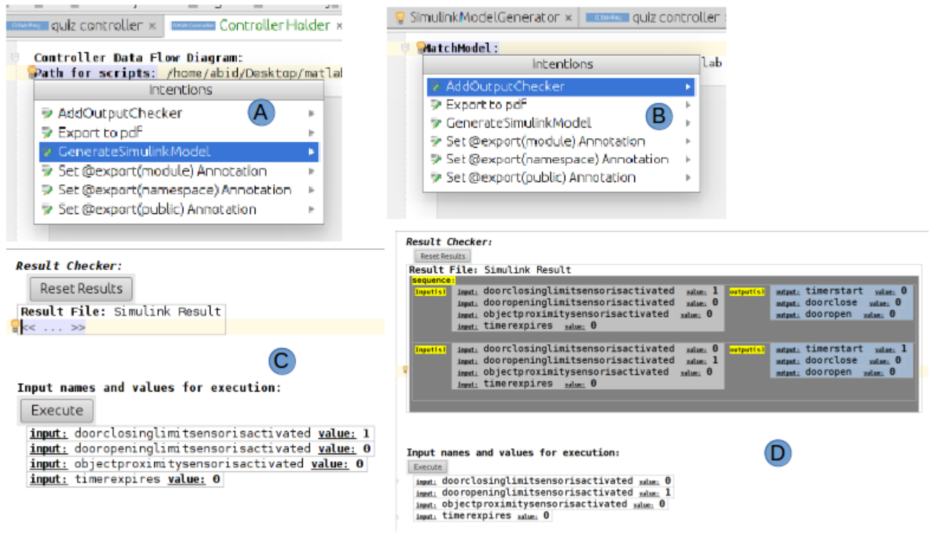

Intentions
Set @export(module) Annotation
Set @export(namespace) Annotation
Set @export(public) Annotation
Transform
A
B
C

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

Req1 : While indicator h                                    r is off , when pupil button 0 is pressed the
    quiz controller shal
Req2 : While indicator h                                    r is off , when pupil button 1 is pressed the
    quiz controller shal
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 .
```

```
Ctrl_0.dc -> Resindicatorhighschoolturnoff.dc0
Ctrl_0.output -> Resindicatorhighschoolturnoff.input0
Ctrl_0.dc -> Resindicatorpupilturnoff.dc0
Ctrl_0.output -> Resindicatorpupilturnoff.input0
Ctrl_0.output -> Resindicatorprofessorturnoff.input0
GIresetbuttonispressed -> Ctrl_0.input
Ctrl_0.dc -> Resindicatorprofessorturnoff.dc0

Global Input
resetbuttonispressed
IfTBActor
Ctrl_0  Def
ResActor
Resindicatorhighschoolturnoff  Def
ResActor
Resindicatorpupilturnoff  Def
ResActor
Resindicatorprofessorturnoff  Def
```

```
Gate Descriptors

Ctrl_0
node IfTB (input: bool) returns (output: int);let output = if input then 1    else 2;tel

Resindicatorhighschoolturnoff
node Res1 (input0: int; A: bool) returns (output: bool);letoutput = if input0 = 1  then true
```

Fig. 4. Controller generation steps: (*A*) Applying Intention (Alt+Enter), (*B*) Data Flow Diagram of the controller (an excerpt) and (*C*) Pseudo code representing the behavior of the blocks (an excerpt)



```
Controller Data Flow Diagram:
Path for scripts:  /home/abid/Desktop/matlal

Result Checker:
Result File: Simulink Result
<< ... >>

Input names and values for execution:
input: doorclosinglimitsensorisactivated value: 1
input: dooropeninglimitsensorisactivated value: 0
input: objectproximitysensorisactivated value: 0
input: timerexpires value: 0
```

```
MatchModel:                                        lab

Result Checker:
Result File: Simulink Result
sequence:
Input(s)  input: doorclosinglimitsensorisactivated  value: 1   output(s)  output: timerstart  value: 0
          input: dooropeninglimitsensorisactivated  value: 0              output: doorclose   value: 0
          input: objectproximitysensorisactivated  value: 0               output: dooropen    value: 0
          input: timerexpires  value: 0
Input(s)  input: doorclosinglimitsensorisactivated  value: 0   output(s)  output: timerstart  value: 1
          input: dooropeninglimitsensorisactivated  value: 1              output: doorclose   value: 0
          input: objectproximitysensorisactivated  value: 0               output: dooropen    value: 0
          input: timerexpires  value: 0

Input names and values for execution:
input: doorclosinglimitsensorisactivated value: 0
input: dooropeninglimitsensorisactivated value: 1
input: objectproximitysensorisactivated value: 0
input: timerexpires value: 0
```

Fig. 5. Simulation steps for verification: (*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 generation, 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) .

Acknowledgements

This work presented in this paper was developed in the context of 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/saadbinabid1/EARS-CTRLReqAnalysis/`.
2. Mathworks tool suite. `https://de.mathworks.com/`.
3. 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.