Formal Modelling of Cruise Control System Using Event-B and Rodin Platform

Sorina Predut, Felician Campean, Marian Gheorghe, Florentin Ipate

UNIVERSITY OF BUCHAREST
VIRTUTE ET SAPIENTIA

UNIVERSITY of BRADFORD

Outline

- Motivation
- Case study: Cruise Control System of an e-Bike
  - Event-B Model for e-Bike Cruise Control
  - Modelling Using iUML-B
  - Validating using ProB plug-in
- Conclusions

Motivation

- ➢ Interdisciplinary system development challenges: design and analysis of **new features** along with already existing and embedded features
- ➢ Design impact on the final product or embedded software
- ➢ **Aim:** combining engineering design analysis with formal methods for system verification and model checking, within a systems engineering environment
    - ➢ Guaranteeing that **the models meet the requirements**
    - ➢ Early detection of misbehaviour and software flaws
- ➢ Approach proposed: integrating various notations utilised in the functional design of complex systems with formal verification (model checking)

Motivation - cont.

- **Event-B**: a formal method for system development
- Main features include the use of **refinement**
- An Event-B model contains 2 parts: **contexts** and **machines**
  - Contexts contain **carrier sets**, **constants**, and **axioms**
  - Machines contain **variables, invariants**, and **events**
- A **machine in Event-B** corresponds to a transition system where variables represent the states and events specify the transitions

Motivation - cont.

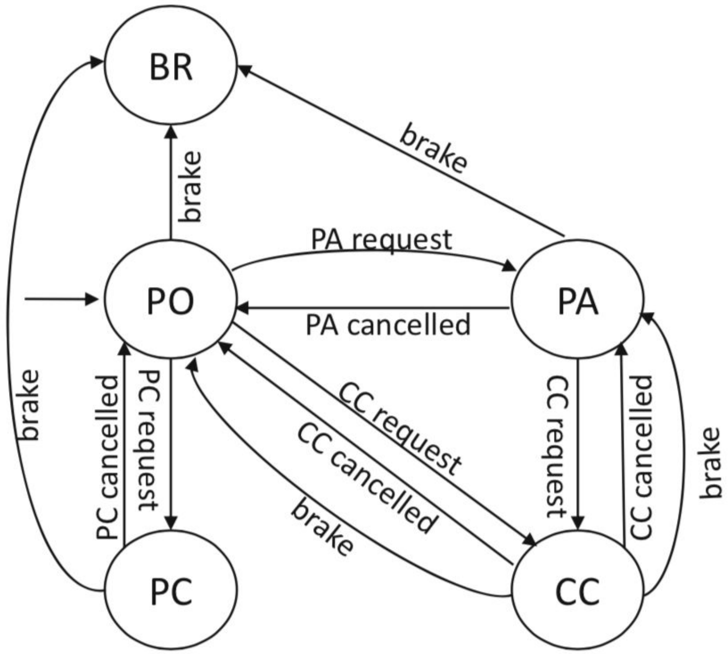
- Event-B is supported by the Rodin Platform
- **Rodin** is an extensible toolkit which includes facilities for:
  - Modelling
  - Verifying about the consistency of models
  - Validating models

Case Study: Cruise Control System of an e-Bike

- Advanced Driver Assist System – ADAS
- Based on brushless hub motor that has the capability to work as a generator as well as a motor
- The propulsion system combines an Electric Drive System (EDS) with a conventional pedal drive
- **Cruise control:** an ADAS technology that automatically controls the speed of a transportation system set by the user
- To deliver the CC feature: EDS adjusts the torque input such that the velocity of the bicycle is maintained, regardless of the effort input from the rider.

State Machine Representation

- Pedal Only - PO: pedal bike
- Pedal Assist – PA: pedal bike with power assistance
- Cruise Control – CC: maintain constant speed
- Pedal Charge – PC: pedal to charge battery
- Brake - Br



Modelling & Verification

- Capturing system functional requirements, use cases, scenarios
- Design a high level prototype of the system
- Model: can be used for simulation, verification (model checking)
- System requirements -> properties to be checked, e.g.
  - The user should be able to request / activate CC from PO (pedal only) or PA (pedal assist)
  - The system should normally return to the state from where it was activated – i.e. PO or PA, respectively
  - The system should not transit from CC to the output state directly; e.g. when the user brakes, the system returns to PA/PO before jumping to the output state

Modelling & Verification - cont.

- ➢ Simulation and testing can only analyse a limited subset of all possible behaviours, and hence they cannot provide an assurance that the system in question works correctly and any undesired behaviour will not happen
- ➢ **Model checking:** receives a mathematical model of the system and a requirement, expressed in a suitable formal logic, and checks if this is verified by exhaustively exploring all system behaviours
- ➢ Model checkers: provide an answer ‘true’ / ‘false’ + counter example
- ➢ This shows some possible behaviour of the system that does not satisfy the given requirement.
- ➢ It allows the designer/modeller to change the system accordingly to fix the error.

Event-B Model for e-Bike Cruise Control

```
MACHINE M0
SEES c_status
VARIABLES
        status
        beforecc
        engrun
INVARIANTS
        inv1:  $status \subseteq STATUS$
        inv3:  $beforecc \subseteq \{PO, PA, UNDEFINED\}$
        inv4:  $engrun \in BOOL$
EVENTS
Initialisation
      begin
            act1: $status := \{PO\}$
            act2: $beforecc := \{UNDEFINED\}$
            act3: $engrun := FALSE$
      end
Event PedalOnly ⟨ordinary⟩ $\widehat{=}$
      when
            grd1:
                $status = \{PA\} \lor status = \{PC\} \lor$
                $(status = \{CC\} \land beforecc = \{PO\})$
      then
            act1: $status := \{PO\}$
            act2: $engrun := FALSE$
      end
```

...

```
Event PedalOnly2CruiseControl ⟨ordinary⟩ $\widehat{=}$
      when
            grd1:  $status = \{PO\}$
      then
            act1: $status := \{CC\}$
            act2: $beforecc := \{PO\}$
            act3: $engrun := TRUE$
      end
```

...

```
Event BrakeCruiseControl2PedalOnly ⟨ordinary⟩ $\widehat{=}$
      when
            grd1:  $status = \{CC\} \land beforecc = \{PO\}$
      then
            act1: $status := \{PO\}$
            act2: $engrun := FALSE$
      end
```

Refinement of M0

```
MACHINE M1
REFINES M0
SEES c_status,c_user_action
VARIABLES
    status
    beforecc
    engrun
    useraction
INVARIANTS
    inv1: $useraction \in STATUS \twoheadrightarrow USER\_ACTION$
EVENTS
Initialisation
    begin
        act1: $status := \{PO\}$
        act2: $beforecc := \{UNDEFINED\}$
        act3: $engrun := FALSE$
        act4: $useraction :\in \{\{PO \mapsto pc\}, \{PO \mapsto pa\}, \{PO \mapsto cc\}\}$
    end
```

```
Event PedalOnly ⟨ordinary⟩ $\widehat{=}$
refines PedalOnly
    when
        grd1:
            $status = \{PA\} \lor status = \{PC\} \lor$
            $(status = \{CC\} \land beforecc = \{PO\})$
            status∈ ℙ(STATUS)\{{PO},{BRAKE},{UNDEFINED}}
    then
        act1: $status := \{PO\}$
        act2: $engrun := FALSE$
        act3: $useraction := \{PA \mapsto pac, CC \mapsto ccc, PC \mapsto pcc\}$
    end
```

...

```
Event PedalOnly2CruiseControl ⟨ordinary⟩ $\widehat{=}$
refines PedalOnly2CruiseControl
    any
        s
    where
        grd1: $s = PO$
        grd2: $status \in \{\{PO\}\}$
    then
        act1: $status := \{CC\}$
        act2: $beforecc := \{PO\}$
        act3: $engrun := TRUE$
        act4: $useraction(s) := cc$
    end
```

```
Event BrakeCruiseControl2PedalOnly ⟨ordinary⟩ $\widehat{=}$
refines BrakeCruiseControl2PedalOnly
    any
        s
    where
        grd1: $s = CC \land beforecc = \{PO\}$
        grd2: $status \in \{\{CC\}\}$
    then
        act1: $status := \{PO\}$
        act2: $engrun := FALSE$
        act3: $useraction(s) := br$
    end
```

Refinement of M1

```
MACHINE M2
REFINES M0
SEES c_status
VARIABLES
      status
      beforecc
      engrun
      brkLvr
INVARIANTS
      inv1: $brkLvr \in BOOL$
EVENTS
Initialisation ⟨extended⟩
    begin
        act1: $status := \{PO\}$
        act2: $beforecc := \{UNDEFINED\}$
        act3: $engrun := FALSE$
        act4: $brkLvr := FALSE$
    end
Event PedalOnly ⟨ordinary⟩ $\widehat{=}$
extends PedalOnly
    when
        grd1:
          $status = \{PA\} \lor status = \{PC\} \lor$
          $(status = \{CC\} \land beforecc = \{PO\})$
    then
        act1: $status := \{PO\}$
        act2: $engrun := FALSE$
    end
```

...

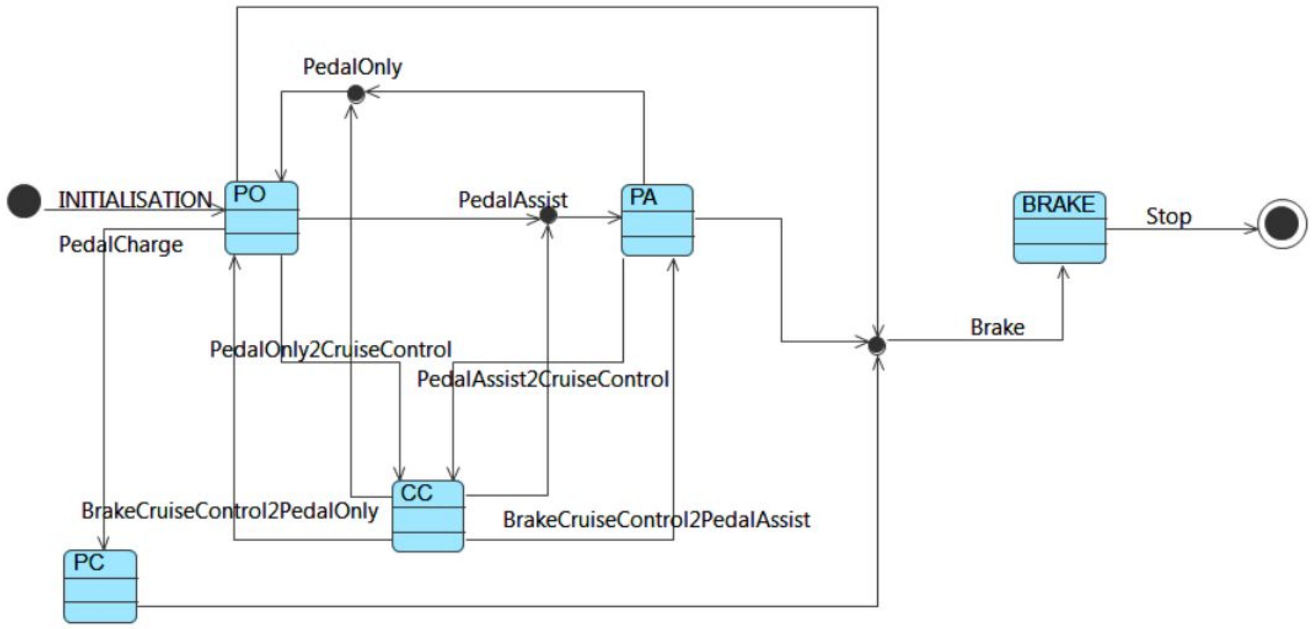
```
Event PressBrkLvr_1 ⟨ordinary⟩ $\widehat{=}$
extends Brake
    when
        grd1: $status = \{PO\} \lor status = \{PA\} \lor status = \{PC\}$
    then
        act1: $status := \{BRAKE\}$
        act2: $engrun := FALSE$
        act3: $brkLvr := FALSE$
    end
Event PressBrkLvr_3 ⟨ordinary⟩ $\widehat{=}$
extends BrakeCruiseControl2PedalOnly
    when
        grd1: $status = \{CC\} \land beforecc = \{PO\}$
    then
        act1: $status := \{PO\}$
        act2: $engrun := FALSE$
        act3: $brkLvr := TRUE$
    end
```

```
Event PressBrkLvr_2 ⟨ordinary⟩ $\widehat{=}$
extends BrakeCruiseControl2PedalAssist
    when
        grd1: $status = \{CC\} \land beforecc = \{PA\}$
    then
        act1: $status := \{PA\}$
        act2: $engrun := TRUE$
        act3: $brkLvr := TRUE$
    end
Event StopBrkLvr ⟨ordinary⟩ $\widehat{=}$
    when
        grd1: $brkLvr = TRUE$
    then
        act1: $brkLvr := FALSE$
    end
END
```

Modelling Using iUML-B

- **iUML** provides a diagrams to help visualise models
- A state-machine will automatically generate the Event-B data elements to implement the states
- Event-B events are expected to already exist to reprezent the transitions
- A choice of 2 **alternative translation encodings** are supported by the iUML tools:
  - state-machines
  - class diagrams
- For the e-Bike we use state-machine diagrams. There are **2 choices of translation**:
  - enumeration
  - variable

State machine diagram



Generated code using boolean variables

```
MACHINE M0
SEES cruise_control
VARIABLES
        PO
        PA
        BRAKE
        CC
        PC
        engrun
        beforecc
INVARIANTS
        typeof_PO:  PO ∈ BOOL
        typeof_PA:  PA ∈ BOOL
        typeof_BRAKE:  BRAKE ∈ BOOL
        typeof_CC:  CC ∈ BOOL
        typeof_PC:  PC ∈ BOOL
        distinct_states_in_iUML: TRUE ∈ {PO, PA, BRAKE, CC, PC}⇒partition({TRUE}, {PO}∩{TRUE}, {PA}∩
            {TRUE}, {BRAKE} ∩ {TRUE}, {CC} ∩ {TRUE}, {PC} ∩ {TRUE})
        inv1:  engrun ∈ BOOL
        inv2:  beforecc ⊆ BEFORECC
```

```
EVENTS
Initialisation
      begin
            init_PO: PO := TRUE
            init_PA: PA := FALSE
            init_BRAKE: BRAKE := FALSE
            init_CC: CC := FALSE
            init_PC: PC := FALSE
            act1: engrun := FALSE
            act2: beforecc := {undefined}
      end
Event  PedalOnly ⟨ordinary⟩ ≙
      when
            isin_PA_or_isin_CC:  (PA = TRUE ∨ CC = TRUE)
      then
            leave_PA: PA := FALSE
            leave_CC: CC := FALSE
            enter_PO: PO := TRUE
            act1: engrun := FALSE
      end
```

...

Validating using ProB plug-in

- **ProB** is an animation and model checking tool which accepts B-models, but is also integrated within the Rodin platform
- Unlike, most model checking tools, ProB works on higher-level formalisms and so it enables a more convenient modeling.
- The animation facilities allow: to visualize, at any moment, the state space, to execute a given number of operations, to see the shortest trace to current state.
- **Properties** that are intended to be verified can be formulated using the **LTL** or the **CTL** formalism.

Validating using ProB plug-in - cont.

- Some examples of LTL operators used in the e-Bike:
  - Globally (G): G p meaning that the property p holds in any state
  - NeXt(X): X p meaning that p holds in the next state
  - Implies (=>), and (&), or (or), negation (not)

Verified properties

Prop. ID	**Property** *Informal query* Formal query (LTL and CTL)	**Result (True / False)**
P1	***The user should be able to request / activate Cruise Control only from PO or PA*** G ({status = {PC} or status = {BRAKE}} => not X {status = {CC}})	true
P2	***The system should not transit directly from CC to brake directly*** G ({status = {CC} & useraction = {CC □→ br}}=> not X {status = {BRAKE}})	true
P3	***When brake is requested in CC the system returns to PA or PO*** G ({status = {CC} & useraction = {CC □→ br}}=> X {status = {PO} or status = {PA}})	true

Verified properties - cont.

Prop. ID	Property *Informal query* Formal query (LTL and CTL)	Result (True / False)
P4	***When system is in CC, PA or PC state, the Engine is running (engrun is True)*** G ({status = {CC} or status = {PA} or status = {PC}} => {engrun = TRUE})	true
P5	***When system is in PO or Brake state, engrun is False*** G ({status = {BRAKE} or status = {PO}}=> {engrun = FALSE})	true
P6	***PA and PC cannot directly activate each other*** G ({status = {PA}} => not X {status = {PC}}) & G({status = {PC}} => not X {status = {PA}})	true

Verified properties - cont.

Prop. ID	**Property** *Informal query* Formal query (LTL and CTL)	**Result (True / False)**
P7	***CC and PC cannot directly activate each other*** G ({status = {CC}} => not X {status = {PC}}) & G({status = {PC}} => not X {status = {CC}})	true
P8	***CC can be activated from a state other than PO or PA*** not {status = {PO} or status = {PA}}U {status = {CC}}	false
P9	***PC can be activated from a state other than PO*** not {status = {PO}} U {status = {PC}}	false

Rodin code

- We open-sourced our implementations at: https://github.com/sinapredut/eBike and https://github.com/sinapredut/eBikeiUMLvar_v2

Conclusions

- ➢ Initial approach towards an integrated methodology to verify the desired behaviours of engineered systems
- ➢ Case study: cruise control system of an e-Bike
- ➢ Modelling an e-Bike cruise control system using Rodin platform and validating its behaviour using the ProB tool
- ➢ Generating a formal model using iUML plug-in

Conclusions

- ➢ Investigation on the Rodin capabilities in case of a continuous domain model of the environment: modelling the continous parts of the system using the plug-in Theory integrated within the Rodin platform
- ➢ **Future work**: make a co-simulation of the closed-loop parts of the controller with a continuous domain model of the environment using MultiSim plug-in as we already have the model implemented in Python

Questions & Answers

Thank you!