Evaluation of a Guideline by Formal Modelling of a Cruise Control System in Event-B

Sanaz Yeganefard, Michael Butler and Abdolbaghi Rezazadeh

users.ecs.soton.ac.uk/mjb
www.event-b.org
www.deploy-project.eu

School of Electronics and Computer Science
University of Southampton, UK

NFM, 13-15 April 2010
Overview

• Managing complexity
  – Abstraction and refinement
  – Event-B and Rodin
  – Sources of system complexity

• Outline of a “Cookbook” for abstraction and refinement

• Applying “cookbook” to Cruise Control System
  – Initial Model
  – Six level of Refinement

• Evaluation and Future Work
Abstraction

• Abstraction can be viewed as a process of simplifying our understanding of a system.

• The simplification should
  – focus on the intended purpose of the system
  – ignore details of how that purpose is achieved.

• The modeller should make judgements about what they believe to be the key features of the system.

• Working with system level reasoning:
  – Involves abstractions of overall system not just software components
  – Emphasise left hand of V process
Refinement

• Refinement is a process of enriching or modifying a model in order to
  1. augment the functionality being modelled, or
  2. explain how some purpose is achieved

• We can perform a series of refinement steps to produce a series of models M1, M2, M3, ...

• Consistency of a refinement:
  – We use proof to verify the consistency of a refinement step
  – Failing proof can help us identify inconsistencies in a refinement step
Event-B (Abrial)

• State-transition model (like ASM, B, VDM, Z)
  – set theory as mathematical language
• Refinement (based on action systems by Back)
  – events: guarded actions
  – data refinement
  – one-to-many event refinement
  – new events (stuttering steps)
• Proof method
  – Refinement proof obligations (POs) generated from models
  – Automated and interactive provers for POs
Rodin Open Tool Platform

• Extension of Eclipse IDE

• Open source development

• Rodin Builder manages:
  – Well-formedness + type checker
  – Consistency/refinement Proof Obligation generator
  – Proof manager: automated and interactive proof
  – Propagation of changes

• Extension points supports plug-ins
  – model-checking, simulation, code generation, UML-B,...

www.event-b.org
Sources of System Complexity

• control laws
  – change acceleration to maintain speed, ...
• operator commands
  – change target speed, suspend, resume, ...
• operator interface
  – buttons, pedals, gearstick ...
• interaction with other features
  – engine management, braking, gearbox,...
• faults and fault management
  – sensor faults, actuator faults, etc, ...
• architecture
  – multi-tasking, distribution, bus, signal evaluation, sensors, actuation, ...

• Where to start modelling?
• What is the right abstraction?
• How do we treat various sources of complexity?
“Cookbook” for control systems (Butler)

• **Guidelines** for abstraction and refinement of control systems in Event-B
• **Influenced** by Parnas 4-variable model
• Abstract models focus on environment phenomenon
• Central role of **system operator** (e.g., driver) is addressed
• **Refinement patterns** for introducing
  – sensing
  – actuation
  – command activation
Four-variable model (Parnas)

• Environment variables
  – Monitored variables \( (speed) \)
  – Controlled variables \( (acceleration) \)

• Controller variables
  – Input variables \( (sensed\ speed) \)
  – Output variables \( (accelerator \ actuation \ value) \)
Requirements

• **NAT** (for nature)
  – describes how monitored variables are influenced by controlled variables (*assumptions*)

• **REQ**
  – describes required values of controlled variables in response to monitored variables (*guarantees*)
In design, we introduce

• **IN**
  – relates monitored variables to input variables

• **OUT**
  – relates output variables to controlled variables
Patterns

• Modelling patterns
  – *Autonomous* controller \((NAT \ and \ REQ)\)
  – *Commanded* controller

• Refinement patterns
  – Separate control and actuation \((OUT)\)
  – Separate sensing and control \((IN)\)
  – Introduce command activation
Autonomous controller model

• Variables
  – *Monitored* variables
  – *Controlled* variables

• Events
  – *Plant* events: modify the *monitored* variables (NAT)
  – *Control* events: modify the *controlled* variables (REQ)
Commanded Controller Model

• **Commanded variable**: value determined by operator (e.g., target speed, cruise status)

• **Command**: modify a commanded variable (e.g., tip-up, switch-off)

• Extension of autonomous model with
  – **Commanded variables** (cmv): can influence control events
  – **Command events** (CMD): modify commanded variables
Applying modelling pattern to cruise control

- **Monitored**: speed
- **Controlled**: acceleration
- **Operator**: target speed, status (on, standby, off)

**Feature elaboration** refinements:
- elaborate events for changing target speed
- elaborate events for changing status through acceleration/clutch or braking pedals
- elaborate events for gears and gear change
- clear identification of different cases
Introducing pedals (in more detail)

• Pressing accelerator → temporary suspension of CCS.
• Releasing accelerator → CCS regain the control of car speed.
• Pressing brake or clutch → permanent suspension.
• Driver can suspend CCS to regain the control of car speed.
Diagrammatic representation (using Jackson Problem Frames)

Autonomous Controller

Commanded Controller
Refinement pattern I: separate control and actuation (OUT)

Event refinement:
Refinement pattern I

• For controlled variable \( c \), introduce *actuation* variable \( c_a \)

• Abstract control events: CTL

\[
\text{CTL} = \quad c := E(m,c)
\]

• Refined events

\[
\begin{align*}
\text{DCN} &= \quad c_a := E(m, c_a) \quad /* \text{refines skip} */ \\
\text{ACT} &= \quad c := c_a \quad /* \text{refines CTL} */ 
\end{align*}
\]
Refinement pattern 1b

• More generally, CTL will have several cases:

\[
\text{CTL}_i = \text{when } G_i(m,c) \text{ then } c := E_i(m,c) \text{ end}
\]

• Cases will be in the refined decision events:

\[
\text{DCN}_i = \text{when } G_i(m, c_a) \text{ then } c_a := E_i(m, c_a) \text{ end}
\]
\[
\text{ACT} = c := c_a \text{ /* refines merge of all CTL}_i */
\]
Refinement pattern II: separate sensing and control (IN)
Refinement pattern II

• For monitored variable \( m \), introduce *sensed* variable \( m_s \)

• Abstract decision events:
  \[
  DCN_i = \text{when } G_i(m, c_a) \text{ then } c_a := E_i(m, c_a) \text{ end}
  \]

• Refined events
  \[
  SEN = m_s := m
  \]
  \[
  DCN_i = \text{when } G_i(m_s, c_a) \text{ then } c_a := E_i(m_s, c_a) \text{ end}
  \]
Refinement pattern III: introduce operator requests

Controller —CMD_RSP— Button —CMD_REQ— Operator

CMD

CMD_REQ  CMD_RSP
Cruise control – applying refinement patterns

1. Introduce actuation
   – distinguish determination of acceleration (internal) from actuation of acceleration (external)

2. Introduce sensing
   – distinguish actual speed from sensed speed stored in controller

3. Introduce buttons
   – Separate operator request for some command from the effect of that command
   – Deal with overloading multiple functions on same button
Evaluation of cookbook

• Identifying monitored, controlled and commanded variables at the abstract level
  – Provides a lot of structure and focus for modeller
• Introducing sensing, actuation and buttons using patterns was straightforward
• Proofs were all automatic
  – because of small refinement steps
  – main proofs: correctness of refinements
• No treatment of feature augmentation in original guideline
• Variable categorisation sometimes fuzzy
  – e.g., gear is monitored from CCS viewpoint, but commanded from a system viewpoint
• Treatment of buttons in original guideline not general enough
Future Work

• **Decomposition to distributed architecture**: separation of the platform, the environment and the software application concepts.
• **Traceability** links between requirements and Event-B models
• **Addressing limitations** of the guidelines
  – timing
  – fault tolerance
  – operator command interface (buttons, pedals, ...)
  – operator display
• Application to other automotive and avionics case studies
Real time...

... or lack of real time

- **Control goal**: maintain vehicle speed within bounds
- **Control strategy**: sample speed periodically and adjust acceleration according to some control laws

- We focus on modelling and refining *strategy* and also dealing with *operator interactions*
  - for this we don’t need real-time, only event ordering

- Our experience with CCS is that operator interaction is a major source of complexity
  - *it is all discrete so is easily dealt with using Event-B*

- Verifying the strategy satisfies the goal does require real-time