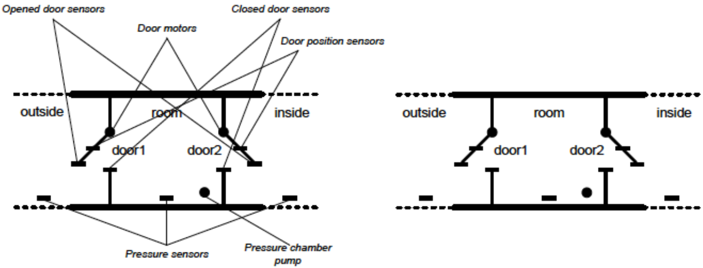
Lecture 3: Outline

- Control systems
- Case study: the sluice system
- Model refinement: simple case
- Relations (introduction)
- Examples

Control systems

- We will look into different types of computer-based systems: how we can model them, express and verify their essential properties
- Control systems: the systems that manage, regulate and command the behaviour of some physical devices or processes
- Typically, they are based on feedback loop, monitoring the state of physical devices or processes (via sensors) and reacting, if necessary, to the state changes by making appropriate commands/signals (to the controlled devices – actuators)
- Control systems often belong to the class of safety-critical systems
- Thus, safety properties (safety invariants) are essential and should be verified

Case study: sluice gate control system



- Sluice connects areas with dramatically different pressures;
- It is unsafe to open a door unless the pressure is levelled between the connected areas;
- The purpose of the system is to operate doors safely by adjusting the pressure in the room.

Example: sluice gate system requirements

1. The purpose of the system is to allow a user to safely travel between inside or outside areas;
2. The system has three locations - outside, middle and inside;
3. The system has two doors - door 1, connecting the outside and middle areas, and door 2, connecting the middle and inside areas;
4. A pump is located in the middle area;
5. Pressure in the inside area is always PRESSURE_LOW;
6. Pressure in the outside area is always PRESSURE_HIGH;

Example: sluice gate system requirements (cont.)

7. The middle area has a pressure sensor reporting the current pressure;
8. Both doors are equipped with sensors reporting the status of a door;
9. There are two types of sensors for each door : switch-type (binary) and value (in the range 0..100) sensors to indicate the door position;
10. The pump changes the pressure in the middle area;
11. When the pump is set to the mode PUMP_IN, it slowly increases the pressure in the middle area;
12. When the pump is set to the mode PUMP_OUT, it slowly decreases the pressure in the middle area;

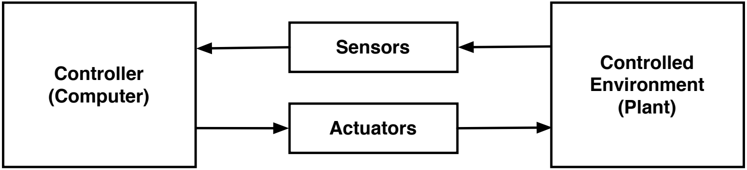
Example: sluice gate system requirements (cont.)

13. When in the mode PUMP_IN mode, the pump automatically stops when the pressure reaches PRESSURE_HIGH;
14. When in the mode PUMP_OUT mode, the pump automatically stops when the pressure reaches PRESSURE_LOW;
15. At most one door is open at any moment;
16. The outside door (door 1) can be opened only when the middle area pressure is PRESSURE_HIGH;
17. The inside door (door 2) can be opened only when the middle area pressure is PRESSURE_LOW;
18. Pressure may only be changed when both doors are closed.

Sluice example: a control system

The sluice system is an instance of a control system.

The general structure of control systems:

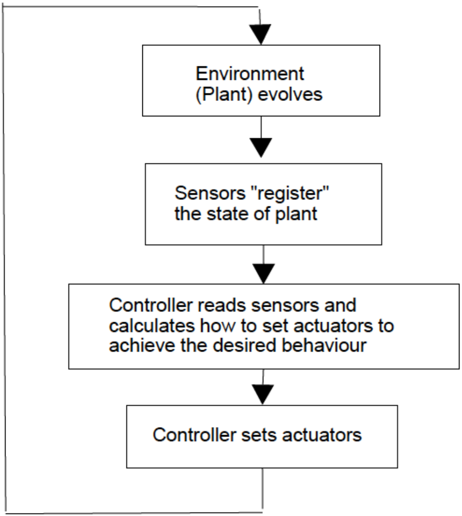


Modelling Control Systems

The control systems are cyclic:

- get inputs from the sensors,
- process them;
- output new values to the actuators.

The overall behaviour of the system is an alternation between the events modelling plant evolution and controller reaction.



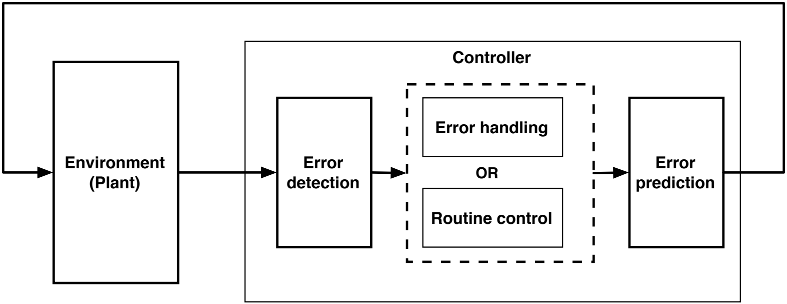
Fault tolerance

- Safety cannot be achieved without fault tolerance (FT);
- FT often relies on redundancy of sensors or actuators;
- Main goal of FT: prevent propagation of a fault to system boundaries (and potentially jeopardise safety);
- Steps of fault tolerance: error detection and error recovery;
- General principle of error detection: find a discrepancy between the expected state of a fault-free system and the observed state.

Modelling a controller

To ensure fault tolerance, its cyclic execution is often split into three steps:

- error detection based on read sensor values;
- update of its internal state and decision on possible control actions (based on both the sensor values and error detection results);
- prediction of the expected sensor values for the next cycle.



Modelling and verifying system dependability

- Ensuring dependability of complex control systems is challenging;
- Formal modelling and refinement in Event-B helps to structure complex requirements and develop systems that are correct and safe by construction;
- How to capture the system requirements in a formal model?
- Too much complexity to model everything at once $\Rightarrow$ complexity can be handled by developing the system model at different abstraction levels, starting with the simplest one and then refining it

An abstract model of the sluice system

CONTEXT
c0
SETS
DOOR
PRESSURE
CONSTANTS
Open
Closed
Low
High
AXIOMS
$partition(DOOR, \{Open\}, \{Closed\})$
$partition(PRESSURE, \{Low\}, \{High\})$
END

Event-B conventions

- *partition* is a shorthand definition of a set that can be partitioned into separate, disjoint parts (subsets)
- If the parts (subsets) are singleton sets, e.g., of the form $\{element\}$, such a definition defines of an enumerated set
- For instance, *partition(Set,{element1}, {element2}, {element3})* is a shorthand for the axioms
    - $element1 \in Set$,
    - $element2 \in Set$,
    - $element3 \in Set$,
    - $element1 \neq element2$,
    - $element1 \neq element3$,
    - $element2 \neq element3$,
    - $Set = \{element1, element2, element3\}$.
- In general, any disjoint subsets instead of *{element}*, e.g., *partition(ITEM, Available, Sold)*

An abstract model of the sluice system (cont.)

MACHINE
m0
SEES
c0
VARIABLES
$door1,\ door2,\ pressure$
INVARIANT
$door1 \in DOOR \wedge door2 \in DOOR$
$pressure \in PRESSURE$
$\neg(door1 = Open \wedge door2 = Open)$
$door1 = Open \Rightarrow pressure = High$
$door2 = Open \Rightarrow pressure = Low$
INITIALISATION
$door1,\ door2 := Closed,\ Closed$
$pressure :\in PRESSURE$

Event-B conventions (cont.)

- $x, y, \ldots := Exp_1, Exp_2, \ldots$ stands for a multiple parallel assignment
- The variables $x, y, \ldots$ should be separate (distinct)
- $x \; :\in \; Set$ is a simple case of a non-deterministic assignment, when any value fro the given set can be assigned to the variable
- It is very useful, when we do not know, cannot control, or do not care, which exact resulting value will be used to update the variable

An abstract model of the sluice system (cont.)

EVENTS

$open1 \;=$

  WHEN $door1 = Closed \;\wedge\; door2 = Closed \;\wedge\; pressure = High$

  THEN $door1 := Open$ **END**

$close1 \;=$

  WHEN $door1 = Open$ **THEN** $door1 := Closed$ **END**

$open2 \;=$

  WHEN $door1 = Closed \;\wedge\; door2 = Closed \;\wedge\; pressure = Low$

  THEN $door2 := Open$ **END**

$close2 \;=$

  WHEN $door2 = Open$ **THEN** $door2 := Closed$ **END**

...

An abstract model of the sluice system (cont.)

$\ldots$

$\quad pressure\_low \;=$

$\qquad \textbf{WHEN}\ door1 = Closed \ \wedge\ door2 = Closed$

$\qquad \textbf{THEN}\ pressure := Low\ \textbf{END}$

$\quad pressure\_high \;=$

$\qquad \textbf{WHEN}\ door1 = Closed \ \wedge\ door2 = Closed$

$\qquad \textbf{THEN}\ pressure := High\ \textbf{END}$

END

Refinement in Event-B

- A way to gradually develop formal models, elaborating on missing implementation details
- Allows to model the system at different abstraction levels, handle its complexity, and structure its requirements
- Consistency of model refinements is supported by the Rodin platform

Refinement in Event-B (cont.)

- Defined separately for a context and a machine;
- For a context component, it is called *extension*;
- Context extension allows
  - introducing new data structures (sets and constants), as well as
  - adding more constraints (axioms) for already defined ones.

Refinement in Event-B (cont.)

- For a machine component, there are several possible kinds of refinement:
  - simple extension of an abstract model by new variables and events (*superposition refinement*);
  - constraining the behaviour of an abstract model (*refinement by reducing model non-determinism*);
  - replacing some abstract variables by their concrete counterparts (*data refinement*);
  - a mixture of those.

Superposition refinement

Probably the simplest way to refine a model by

- Adding new variables and events;
- Reading and updating new variables in old event guards and actions;
- Interrelating new and old variables by new invariants.

Restriction: the old variables cannot be updated in new events!

Sluice example: the first refinement

The goal is to abstractly model the feedback loop of a control system. An example of superposition refinement:

- Introduce a new type $PHASE = \{Env, Det, Cont, Pred\}$;
- Add new variables $phase \in PHASE$ and $failure \in BOOL$;
- Introduce new events *Environment*, *Detection* and *Prediction* with the corresponding guards;
- Strengthen guards of the old events, making the events a part of the controller phase;
- Introduce new events *stop* and *other_control*.

The sluice system: a refined model

CONTEXT
  c1
EXTENDS
  c0
SETS
  $PHASE$
CONSTANTS
  Env
  Det
  $Cont$
  $Pred$
AXIOMS
  $partition(PHASE, \{Env\}, \{Det\}, \{Cont\}, \{Pred\})$
END

The sluice system: a refined model (cont.)

MACHINE
m1
REFINES
m0
SEES
c1
VARIABLES
door1, door2, pressure, phase, failure
INVARIANT
$phase \in PHASE \wedge failure \in BOOL$
$failure = TRUE \Rightarrow phase = Cont$
$phase = Pred \Rightarrow failure = FALSE$
$phase = Env \Rightarrow failure = FALSE$
INITIALISATION
...
$phase, failure := Env, FALSE$

The sluice system: a refined model (cont.)

EVENTS

$\mathit{Environment} \;=$

WHEN $\mathit{phase} = \mathit{Env}$ **THEN** $\mathit{phase} := \mathit{Det}$ **END**

$\mathit{Detection} \;=$

WHEN $\mathit{phase} = \mathit{Det}$ **THEN**

$\mathit{failure} :\in \mathit{BOOL}$

$\mathit{phase} := \mathit{Cont}$

END

$\mathit{open1} \;=$

WHEN ... $\mathit{phase} = \mathit{Cont} \wedge \mathit{failure} = \mathit{FALSE}$

THEN ... $\mathit{phase} := \mathit{Pred}$ **END**

$\mathit{close1} \;=$

WHEN ... $\mathit{phase} = \mathit{Cont} \wedge \mathit{failure} = \mathit{FALSE}$

THEN ... $\mathit{phase} := \mathit{Pred}$ **END**

...

The sluice system: a refined model (cont.)

...

<open2, close2, pressure_low, pressure_high modified similarly >

$other\_control \ =$

WHEN $phase = Cont \wedge failure = FALSE$

THEN $phase := Pred$ **END**

$stop \ =$

WHEN $phase = Cont \wedge failure = TRUE$

THEN END

$Prediction \ =$

WHEN $phase = Pred$

THEN $phase := Env$ **END**

END

Sluice example: a refined model (cont.)

- Here, *failure* abstractly models the unrecoverable system failure leading to the shutdown (stop) of the system
- Since the concrete detection mechanisms are still missing, failure detection is modelled non-deterministically as $failure :\in BOOL$
- The controller phase may contain other control actions (e.g., managing the pump or door motors), so we reserve a possibility to add these actions in the abstract event *other_control*
- The event *stop* can be also refined to include concrete system shutdown mechanisms

Sluice example: possible refinement plan

Five small incremental refinement steps:

- Introducing feedback loop of a control system (m1);
- Elaborating on the environment part and adding sensors (m2);
- Data refining failure modes (m3);
- Elaborating on error detection (m4);
- Introducing actuators and refining error prediction (m5).

Relations: introduction

- Modelling in the B Method based on sets – collections of elements of the same underlying type
- Often this is not enough: connections between elements of different types should be expressed
- Relations allow us to express more complicated interconnections and relationships formally
- Relations are often called many-to-many mappings

Relations (cont.)

- A relation R between sets S and T can be represented as a set of pairs (s, t) representing those elements of S and T that are related
- A pair is syntactically represented in Event-B as $(s \mapsto t)$ or (s,, t) in ascii
- Mathematically, a relation between sets S and T is a member of $\mathbb{P}(S \times T)$, i.e., a subset of $S \times T$
- Reminder: $S \times T$ – all possible pairs from S and T
- Shorthand notation: $S \leftrightarrow T \;\equiv\; \mathbb{P}(S \times T)$
- In other words, $R \in S \leftrightarrow T$ is equivalent to
  $R \in \mathbb{P}(S \times T)$ or $R \subseteq S \times T$

Relations (cont.)

- Since a relation is just a special form of a set, all set operations are applicable to relations
- Example: a relation
  $owns\_camera \in PERSON \leftrightarrow CAMERA$
- Initialisation:
  $owns\_camera := \{Jonas \mapsto Canon, Vaidas \mapsto Nikon,$
  $Vaiva \mapsto Sony, Jonas \mapsto Sony, Sandra \mapsto Pentax\}$
- Checking for membership:
  $Jonas \mapsto Sony \in owns\_camera$ (TRUE)
  $Vaiva \mapsto Canon \in owns\_camera$ (FALSE)
- Similarly, $\cup$, $\cap$, $\setminus$, $\subseteq$, *card*, ... on relations

Relation domain and range

- The *domain* of a relation $R \in S \leftrightarrow T$ is the subset of elements of S that are related to something in T
- Relation domain (denoted as $dom(R)$) is defined by $\{x \mid x \in S \wedge \exists y.\ (y \in T \wedge (x, y) \in R)\}$
- Example: $dom(owns\_camera) = \{Jonas, Vaidas, Vaiva, Sandra\}$
- The *range* of a relation $R \in S \leftrightarrow T$ is the subset of elements of T that are related to something in S
- Relation range (denoted as $ran(R)$) is defined by $\{y \mid y \in T \wedge \exists x.\ (x \in S \wedge (x, y) \in R)\}$
- Example: $ran(owns\_camera) = \{Canon, Nikon, Sony, Pentax\}$

Relation filtering operations

Graphical notation, followed by the equivalent ascii notation:

$S \lhd R$	S <| R	domain restriction
$S \unlhd R$	S <<| R	domain subtraction
$R \rhd S$	R |> S	range restriction
$R \unrhd S$	R |>> S	range subtraction

Examples:

$$\{Jonas\} \lhd owns\_camera = \{Jonas \mapsto Canon, Jonas \mapsto Sony\}$$
$$\{Mindaugas\} \lhd owns\_camera = \varnothing$$
$$\{Jonas, Vaiva\} \unlhd owns\_camera = \{Vaidas \mapsto Nikon, Sandra \mapsto Pentax\}$$
$$\{Mindaugas\} \unlhd owns\_camera = owns\_camera$$

$$owns\_camera \rhd \{Sony\} = \{Vaiva \mapsto Sony, Jonas \mapsto Sony\}$$
$$owns\_camera \unrhd \{Sony, Canon, Pentax\} = \{Vaidas \mapsto Nikon\}$$

Homework: a hotel booking system

- The task: to create an Event-B system model within the Rodin platform for the given hotel reservation system requirements (the next two slides)
- The task has to be finished and presented within 3 weeks from today
- A finished Rodin project has to be exported (as a zip file) and submitted as your assignment solution from the course page in Moodle
- Separately, the solution should be personally "defended" during one of exercise sessions

Homework: a hotel booking system (requirements)

1. The hotel booking system handles room reservation by customers;
2. The system must have operations (events) for room reservation, cancellation, customer check-in, and customer check-out;
3. Only vacant hotel rooms can be reserved;
4. A reservation stores the information about the reserved room and the customer;
5. A reservation can be cancelled;
6. After cancellation, the previously reserved room becomes vacant;

Homework: a hotel booking system (requirements)

7. After a customer's check-in, the reserved room gets the status "occupied";
8. Each occupied room is reserved and associated with the same customer;
9. After a customer's check-out, the previously occupied room becomes vacant;
10. A vacant room cannot be at the same time considered as reserved or occupied by the system;
11. Once a reservation is cancelled or a customer checks out, all the information about the reservation is deleted from the system.

Homework: a hotel booking system (cont.)

Hints:

- Decide first on your static data in the context component (e.g., the needed abstract or enumerated sets)
- Model the vacant rooms as a set, while the reserved and occupied rooms as the corresponding relations;
- To remove the information from the relational variables, use the domain or range subtraction operations;
- Express the logical relationships between the vacant, reserved and occupied rooms as the respective system invariants.