15. A Mechanical Press Controller

Jean-Raymond Abrial (ETHZ)

March 2008
1. Informal presentation of the example

2. Presentation of some design patterns

3. Writing the requirement document

4. Proposing a refinement strategy

5. Development of the model using refinements and design patterns

6. Demos
1. **Informal** Presentation of the **Example**
- A mechanical press controller

- Adapted from a real system

- The real system is coming from INRST:

  Institut National de la Recherche sur la Sécurité du Travail
Basic Equipment

- A **Vertical Slide** with a tool at its lower extremity

- An electrical **Rotating Motor**

- A **Rod** connecting the motor to the slide.

- A **Clutch** engaging or disengaging the motor on the rod

- When the clutch is disengaged, **the slide stops “immediately”**
Basic Commands

- Button B1: start motor
- Button B2: stop motor
- Button B3: engage clutch
- Button B4: disengage clutch
Basic User Actions

- Action 1: **Change the tool** at the lower extremity of the slide

- Action 2: **Put a part** to be treated under the slide

- Action 3: **Remove the part**
1. start the motor (button B1)

2. change the tool (action 1)

3. put a part (action 2),

4. engage the clutch (button B3): the press now works,

5. disengage the clutch (button B4): the press does not work,

6. remove the part (action 3),

7. repeat zero or more times steps 3 to 6,

8. repeat zero or more times steps 2 to 7,

9. stop the motor (button B2).
- step 2 (change the tool),

- step 3 (put a part),

- step 6 (remove the part) are all DANGEROUS
Second Schematic View

CONTROLLER EQUIPMENT

COMMANDS

CONTROLLER

EQUIPMENT
More Elaborate Commands for Protecting the User

- Controlling the way the clutch is engaged or disengaged

- Protection by means of a Front Door
The Front Door

open

closed
- Initially, the door is open

- When the user presses button B3 to engage the clutch, the door is first closed BEFORE engaging the clutch

- When the user presses button B4 to disengage the clutch, the door is opened AFTER disengaging the clutch

- Notice: The door has no button.
Summary of Connections
Overview

CLUTCH

CONTROLLER

MOTOR

DOOR

Motor
Start Stop

Clutch
Start Stop

Motor_actuator

Motor_sensor

Clutch_actuator

Clutch_sensor

Door_actuator

Door_sensor

CLUTCH

MOTOR

CONTROLLER

DOOR
2. Presentation of some Design Patterns
Motivations

- A number of similar behaviors

- Some complex situations to handle
- A **specific action** results eventually in having a **specific reaction**:

  - Pushing **button B1** results eventually in **starting the motor**

  - Pushing **button B4** results eventually in **disengaging the clutch**

  - . . .
- Correlating two pieces of equipment:

  - When the **clutch** is engaged then the **motor** must work

  - When the **clutch** is engaged then the **door** must be closed
- Making an action dependent of another one:

- Engaging the clutch implies closing the door first

- Disengaging the clutch means opening the door afterwards
- Here is a sequence of events:

  (1) **User** pushes button B1 (start motor)
  (1’) **User does not remove his finger from button B1**
  (2) **Controller** sends the starting command to the motor
  (3) **Motor** starts and sends feedback to the controller
  (4) **Controller** is aware that the motor works
  (5) **User** pushes button B2 (stop motor)
  (6) **Controller** sends the stop command to the motor
  (7) **Motor** stops and sends feedback to the controller
  (8) **Controller** is aware that the motor does not work
  (9) **Controller must not send the starting command to the motor**
Motivation: Example of Some Complex Situation

- Here is a sequence of events:
  
  (1) **User** pushes button B1 (start motor)
  
  (2) **Controller** sends the starting command to the motor
  
  (3.1) **Motor** starts and sends feedback to the controller
  
  (3.2) **User** pushes button B2 (stop motor)
  
- (3.1) and (3.2) may occur *simultaneously*

- If **controller** treats (3.1) before (3.2): motor is stopped

- If **controller** treats (3.2) before (3.1): motor is not stopped
- We want to build systems which are **correct by construction**

- We want to have **more methods** for doing so

- "**Design pattern**" is an Object Oriented concept

- We would like to **borrow this concept** for doing **formal developments**

- A preliminary tentative with **reactive system** developments

- Advantage: **systematic developments** and also refinement of proofs
- This is an engineering concept

- It can be used outside OO

- The goal of each DP is to solve a certain category of problems

- But the design pattern has to be adapted to the problem at hand

- Is it compatible with formal developments?

- Let’s apply this approach to the design of reactive systems
- A design pattern isn’t a finished design that can be transformed into code

- It is a template for how to solve a problem that can be used in many different situations

- Patterns originated as an architectural concept by Christopher Alexander

- "Design Patterns: Elements of Reusable Object-Oriented Software" published in 1994 (Gamma et al)
- Design pattern can speed up the development process by providing tested and proven development paradigms

- The documentation for a design pattern should contain enough information about the problem that the pattern addresses, the context in which it is used, and the suggested solution.

- Some feel that the need for patterns results from using computer languages or techniques with insufficient abstraction
An Action Pattern
Action and Reaction Patterns
- Sometimes, the reaction has *not enough time* to react

- Because the action moves *too quickly*
- Sometimes, the reaction *always follows* the action

- They are both *synchronized*
- We built first a model of a weak reaction

- The strong reaction will be a refinement of the weak one
Model for weak action and reaction: the State

variables: \( a \), \( r \), \( ca \), \( cr \)

\[
\begin{align*}
\text{pat0}_1: \quad & a \in \{0, 1\} \\
\text{pat0}_2: \quad & r \in \{0, 1\} \\
\text{pat0}_3: \quad & ca \in \mathbb{N} \\
\text{pat0}_4: \quad & cr \in \mathbb{N} \\
\text{pat0}_5: \quad & cr \leq ca
\end{align*}
\]

- \( a \) denotes the action
- \( r \) denotes the reaction
- \( ca \) and \( cr \) denote how many times \( a \) and \( r \) are set to 1
- \text{pat0}_5 formalizes the weak reaction
Model for weak action and reaction: the Events (1)

\[
\begin{align*}
\text{a}_{\text{on}} & \quad \text{when} \quad a = 0 \\
& \quad \text{then} \\
& \quad a := 1 \\
& \quad ca := ca + 1 \\
& \quad \text{end} \\
\text{a}_{\text{off}} & \quad \text{when} \quad a = 1 \\
& \quad \text{then} \\
& \quad a := 0 \\
& \quad \text{end}
\end{align*}
\]
Model for weak action and reaction: the Events (2)

\[\begin{align*}
\text{r}_\text{on} & \\
\text{when} & \\
  r &= 0 \\
  a &= 1 \\
\text{then} & \\
  r &:= 1 \\
  cr &:= cr + 1 \\
\text{end} & \\
\text{r}_\text{off} & \\
\text{when} & \\
  r &= 1 \\
  a &= 0 \\
\text{then} & \\
  r &:= 0 \\
\text{end} &
\end{align*}\]
Summary of Events

\begin{align*}
a_{\text{on}} & \\
& \text{when } a = 0 \\
& \text{then } a := 1 \\
& \text{ca} := \text{ca} + 1 \\
& \text{end} \\
\end{align*}

\begin{align*}
a_{\text{off}} & \\
& \text{when } a = 1 \\
& \text{then } a := 0 \\
& \text{end} \\
\end{align*}

\begin{align*}
r_{\text{on}} & \\
& \text{when } r = 0 \\
& \text{a} = 1 \\
& \text{then } a := 1 \\
& \text{then } r := 1 \\
& \text{cr} := \text{cr} + 1 \\
& \text{end} \\
\end{align*}

\begin{align*}
r_{\text{off}} & \\
& \text{when } r = 1 \\
& \text{a} = 0 \\
& \text{then } r := 0 \\
& \text{end} \\
\end{align*}
Summary of Weak Synchronization

variables: \( a, r, ca, cr \)

\[ \text{pat0}_1: \quad a \in \{0, 1\} \]
\[ \text{pat0}_2: \quad r \in \{0, 1\} \]
\[ \text{pat0}_3: \quad ca \in \mathbb{N} \]
\[ \text{pat0}_4: \quad cr \in \mathbb{N} \]
\[ \text{pat0}_5: \quad cr \leq ca \]

\[ \text{init} \]
\[ a := 0 \]
\[ r := 0 \]
\[ ca := 0 \]
\[ cr := 0 \]

\[ \text{a\_on} \]
\[ \text{when} \]
\[ a = 0 \]
\[ \text{then} \]
\[ a := 1 \]
\[ ca := ca + 1 \]
\[ \text{end} \]

\[ \text{a\_off} \]
\[ \text{when} \]
\[ a = 1 \]
\[ \text{then} \]
\[ a := 0 \]
\[ \text{end} \]

\[ \text{r\_on} \]
\[ \text{when} \]
\[ r = 0 \]
\[ a = 1 \]
\[ \text{then} \]
\[ r := 1 \]
\[ a = 0 \]
\[ \text{then} \]
\[ cr := cr + 1 \]
\[ \text{end} \]

\[ \text{r\_off} \]
\[ \text{when} \]
\[ r = 1 \]
\[ a = 0 \]
\[ \text{then} \]
\[ r := 0 \]
\[ \text{end} \]

Nothing guarantees that the invariants are preserved
Invariant Preservation Statement by an Event

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Guards of the event</th>
<th>INV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>⊢</td>
<td></td>
</tr>
<tr>
<td>⊢ Modified Invariant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is called a Proof Obligation Rule

- The rule takes the form of a sequent

- A sequent is made of:
  - an antecedent containing zero or more assumptions
  - a consequent containing a predicate to prove
Naming the Proof Obligation

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Guards of the event</th>
<th>INV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>⊢</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modified Invariant</td>
<td></td>
</tr>
</tbody>
</table>

This is called a Proof Obligation Rule

- We have 5 invariants: pat0_1 to pat0_5

- We have 4 events: a_on, a_off, r_on, and r_off

- This makes 20 Proof Obligations

- Naming conventions: event-name / invariant-name / INV
## Invariant Preservation Statement by an Event

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Guards of the event</th>
<th>INV</th>
<th>POs are generated by a tool: the POG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>格</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>⊢</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modified Invariant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| a\_on | when | a = 0 | then | a := 1 | ca := ca + 1 | end |

\[
\begin{align*}
    & a \in \{0, 1\} \\
    & r \in \{0, 1\} \\
    & ca \in \mathbb{N} \\
    & cr \in \mathbb{N} \\
    & cr \leq ca \\
    & a = 0 \\
\end{align*}
\]

\[
\begin{align*}
    & 1 \in \{0, 1\} \\
\end{align*}
\]

Preservation of invariant \textbf{pat0\_1} by event \textbf{a\_on}
- The preservation proof of \texttt{pat0\_5} \((cr \leq ca)\) by event \texttt{r\_on} fails

\begin{verbatim}
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end

\ldots

\ldots
\end{verbatim}
- The preservation proof of `pat0_5 (cr ≤ ca)` by event `r_on` fails when `r = 0` and `a = 1` then

\[
\begin{align*}
  & r := 1 \\
  & cr := cr + 1
\end{align*}
\]

- We have to add the assumption `cr < ca` in our sequent
- The preservation proof of pat0_5 (\(cr \leq ca\)) by event r_on fails

\[
\begin{align*}
\text{r_on} & \\
\text{when} & \\
\quad r &= 0 \\
\quad a &= 1 \\
\text{then} & \\
\quad r &= 1 \\
\quad cr &= cr + 1 \\
\text{end} & \\
\end{align*}
\]

\[
\begin{align*}
\ldots & \\
\quad cr &\leq ca \\
\quad r &= 0 \\
\quad a &= 1 \\
\vdash & \\
\quad cr + 1 &\leq ca \\
\ldots & \quad \text{pat0_5} \\
\ldots & \quad \text{guard of r_on} \\
\ldots & \quad \text{guard of r_on} \\
\ldots & \quad \text{modified pat0_5}
\end{align*}
\]

- We have to add the assumption \(cr < ca\) in our sequent

- Two solutions: strengthening the guard or adding a new invariant
- Drawback: One has to keep variables \( cr \) and \( ca \) in the guard
- These variables were introduced for the modelling only
First Solution: Adding an Invariant

- We cannot introduce invariant $cr < ca$ directly (it does not hold)
- We introduce an implicative invariant

\[
\text{pat0\_6: } r = 0 \land a = 1 \implies cr < ca
\]

\[
\begin{align*}
\text{r\_on} \\
\text{when} \\
r = 0 \\
a = 1 \\
\text{then} \\
r := 1 \\
\text{end} \\
\end{align*}
\]

\[
\begin{align*}
\ldots \\
\text{cr} \leq ca \quad \text{pat0\_5} \\
r = 0 \land a = 1 \implies cr < ca \quad \text{pat0\_6} \\
r = 0 \\
a = 1 \\
\vdash cr + 1 \leq ca \\
\ldots \\
\text{guard of } \text{r\_on} \\
\text{guard of } \text{r\_on} \\
\text{modified pat0\_5}
\end{align*}
\]

- Drawback: One has to prove that this new invariant is maintained
Intuition Behind the new Invariant

\[ \text{pat0}_6: \quad r = 0 \land a = 1 \Rightarrow cr < ca \]

ca is incremented

cr is incremented

\[ \begin{align*}
\text{r=0} & \\
\text{a=1} & \\
\text{cr<ca} &
\end{align*} \]
Problems with the New Invariant Preservation

\[ r = 0 \land a = 1 \implies cr < ca \]

- No problem with \texttt{a\_on} since \texttt{ca} is incremented
- No problem with \texttt{a\_off} since \texttt{a} becomes 0
- No problem with \texttt{r\_on} since \texttt{r} becomes 1
- No problem with \texttt{r\_off} since \texttt{a} = 0 (guard)
Example of PO to Prove

\[
\text{pat0.6: } r = 0 \land a = 1 \implies cr < ca
\]

The preservation of this invariant by \texttt{r.on} leads to proving:

\[
\begin{align*}
\text{r.on} \\
\text{when} \\
\quad r = 0 \\
\quad a = 1 \\
\text{then} \\
\quad r := 1 \\
\quad cr := cr + 1 \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
1 = 0 \land a = 1 & \implies cr + 1 < ca \\
\end{align*}
\]

which holds trivially
Summary of the State of the weak Reaction

\[ \begin{align*}
\text{pat0}_1: & \quad a \in \{0, 1\} \\
\text{pat0}_2: & \quad r \in \{0, 1\} \\
\text{pat0}_3: & \quad ca \in \mathbb{N} \\
\text{pat0}_4: & \quad cr \in \mathbb{N} \\
\text{pat0}_5: & \quad cr \leq ca \\
\text{pat0}_6: & \quad r = 0 \land a = 1 \Rightarrow cr < ca
\end{align*} \]
Summary of the Events of the weak Reaction

The counters have been removed

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Action 1</th>
<th>Action 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>$a := 0$</td>
<td>$r := 0$</td>
<td></td>
</tr>
<tr>
<td>a_on</td>
<td>$a = 0$</td>
<td>$a := 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td>r_on</td>
<td>$r = 0$</td>
<td>$a = 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r := 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td>a_off</td>
<td>$a = 1$</td>
<td>$a := 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td>r_off</td>
<td>$r = 1$</td>
<td>$a = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r := 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
Weak Synchronization of Events

\[\text{a}_{\text{on}} \quad \text{a}_{\text{off}}\]

\[\text{r}_{\text{on}} \quad \text{r}_{\text{off}}\]
- We add the following invariant

\[
\text{pat1}_1: \quad ca \leq cr + 1
\]

- Remember invariant \text{pat0}_5

\[
\text{pat0}_5: \quad cr \leq ca
\]

We have thus:

\[
cr \leq ca \leq cr + 1
\]
Problems with the New Invariant Preservation

- Problem with \texttt{a\_on} since \texttt{ca} is incremented
- No problem with \texttt{a\_off} since no changes
- No problem with \texttt{r\_on} since \texttt{cr} is incremented
- No problem with \texttt{r\_off} since no changes
- Event a_on cannot maintain pat1_1

<table>
<thead>
<tr>
<th>Event a_on</th>
</tr>
</thead>
<tbody>
<tr>
<td>when a = 0</td>
</tr>
<tr>
<td>then a := 1</td>
</tr>
<tr>
<td>ca := ca + 1</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
    cr & \leq ca \\
    ca & \leq cr + 1 \\
    a & = 0 \\
\end{align*}
\]

\(\vdash ca + 1 \leq cr + 1\)

pat0_5
pat1_1

Guard of a_on

modified pat1_1
- Event a_on cannot maintain \texttt{pat1.1}

```
\begin{array}{ll}
  \text{a\_on} \\
  \text{  when} \\
  \qquad a = 0 \\
  \text{  then} \\
  \qquad a := 1 \\
  \qquad ca := ca + 1 \\
  \text{end}
\end{array}
```

\[
\begin{align*}
  cr & \leq ca \\
  ca & \leq cr + 1 \\
  a & = 0 \\
  \Gamma & \vdash ca + 1 \leq cr + 1 \\
\end{align*}
\]

- We need the assumption \( ca \leq cr \)
- Event \texttt{a\_on} cannot maintain \texttt{pat1\_1}

\begin{verbatim}
\begin{verbatim}
\texttt{a\_on}
\texttt{\ when}
\texttt{\ \ a = 0}
\texttt{\ then}
\texttt{\ \ a := 1}
\texttt{\ \ ca := ca + 1}
\texttt{\ end}
\end{verbatim}

\begin{align*}
\texttt{cr} & \leq \texttt{ca} \\
\texttt{ca} & \leq \texttt{cr} + 1 \\
\texttt{a} & = 0 \\
\end{align*}

\text{guard of \texttt{a\_on}} \quad \text{modified \texttt{pat1\_1}}

- We need the assumption \texttt{ca} \leq \texttt{cr}

- But we already have assumption \texttt{cr} \leq \texttt{ca} (this is \texttt{pat0\_5})
- Event $a_{on}$ cannot maintain $\text{pat1.1}$

$$\begin{align*}
\text{a}_{\text{on}} \\
\text{when} & \quad a = 0 \\
\text{then} & \quad a := 1 \\
& \quad ca := ca + 1 \\
\text{end}
\end{align*}$$

$$\begin{align*}
cr & \leq ca \\
ca & \leq cr + 1 \\
a & = 0
\end{align*}$$

- We need the assumption $ca \leq cr$
- But we already have assumption $cr \leq ca$ (this is $\text{pat0.5}$)
- Thus we need the assumption $cr = ca$
- Event `a_on` cannot maintain `pat1_1`

- We need the assumption `ca ≤ cr`
- But we already have assumption `cr ≤ ca` (this is `pat0_5`)
- Thus we need the assumption `cr = ca`
- This suggests the new invariant: `a = 0 ⇒ cr = ca`
Problems with the New Invariant Preservation

\[ \text{pat1\_2: } a = 0 \Rightarrow cr = ca \]

- No problem with \texttt{a\_on} since \( a \) becomes 1
- Problem with \texttt{a\_off} since \( a \) becomes 0
- No problem with \texttt{r\_on} since \( a = 1 \) (guard)
- No problem with \texttt{r\_off} since no changes
Proposing a new invariant

\[
\text{pat1\_2:} \quad a = 0 \implies cr = ca
\]

- The proof of maintenance of this invariant by a\_off fails

\[
\begin{align*}
\text{a\_off} \\
\quad \text{when} & \quad a = 1 \\
\quad \text{then} & \quad a := 0 \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
a &= 0 \implies cr = ca \\
a &= 1 \\
\vdash & \\
0 &= 0 \implies cr = ca
\end{align*}
\]

- This suggests a new invariant:

\[
a = 1 \implies cr = ca
\]
We need: \( a = 1 \implies cr = ca \)

- But we already have the following:

\[
\text{pat0}_6: \quad a = 1 \land r = 0 \implies cr < ca
\]

This suggests the following:

\[
\text{pat1}_3: \quad a = 1 \land r = 1 \implies cr = ca
\]
- In order for \( a_{\text{off}} \) to prove \( \text{pat1}_2 \) \((a = 0 \Rightarrow cr = ca)\)

\begin{align*}
\text{a\_off} \\
\text{when} \quad a = 1 \\
\text{then} \quad a := 0 \\
\text{end}
\end{align*}

\begin{align*}
a = 0 & \Rightarrow cr = ca \quad & \text{pat1}_2 \\
a = 1 \land r = 1 & \Rightarrow cr = ca \quad & \text{pat1}_3 \\
\end{align*}

- We need to strengthen its guard because of \( \text{pat1}_3 \):
- In order for \texttt{a\_off} to prove \texttt{pat1\_2} \((a = 0 \Rightarrow cr = ca)\)

\begin{verbatim}
\begin{tabular}{c|c}
\texttt{a\_off} & \texttt{pat1\_2} \((a = 0 \Rightarrow cr = ca)\) \texttt{pat1\_3} \\
\hline
\texttt{when} & \texttt{pat1\_3} \\
\texttt{r = 1} & \texttt{new guard} \\
\texttt{a = 1} & \texttt{guard} \\
\texttt{then} & \texttt{modified pat1\_2} \\
\texttt{a := 0} & \\
\texttt{end} & \\
\end{tabular}
\end{verbatim}

- We need to strengthen its guard because of \texttt{pat1\_3}: 

\(0 = 0 \Rightarrow cr = ca\)
Problems with the New Invariant Preservation

\[ \text{pat1.3: } a = 1 \land r = 1 \Rightarrow cr = ca \]

- Problem with \texttt{a.on} since \(a\) becomes 1 and \(ca\) is incremented
- No problem with \texttt{a.off} since \(a\) becomes 0
- Problem with \texttt{r.on} since \(a = 1\) (guard), \(r\) becomes 1, and \(cr\) is incremented
- No problem with \texttt{r.off} since \(r\) becomes 0
- Event $a\_on$ cannot maintain invariant $\text{pat1}_3$

$$ a = 1 \land r = 1 \Rightarrow cr = ca $$

```
\text{a\_on}
  \text{when}
    \begin{align*}
      a &= 0 \\
    \end{align*}
  \text{then}
    \begin{align*}
      a &:= 1 \\
      ca &:= ca + 1 \\
    \end{align*}
\text{end}
```

- This suggest strengthening the guard of $a\_on$: $r = 0$
- Event a_on cannot maintain invariant \textbf{pat1.3}

\[ a = 1 \land r = 1 \Rightarrow cr = ca \]

- This suggest strengthening the guard of a_on: \( r = 0 \)
Problems with the New Invariant Preservation

\[
\text{pat1.3: } a = 1 \land r = 1 \Rightarrow cr = ca
\]

- No problem with \texttt{a\_on} since \( r = 0 \) (guard)
- No problem with \texttt{a\_off} since \( a \) becomes 0
- Problem with \texttt{r\_on} since \( a = 1 \) (guard), \( r \) becomes 1, and \( cr \) is incremented
- No problem with \texttt{r\_off} since \( r \) becomes 0
What about Event $r\_on$ and Invariant $\text{pat1}_3$

- Invariant $\text{pat1}_3$

\[
\begin{align*}
a &= 1 \land r = 1 & \Rightarrow & & cr = ca
\end{align*}
\]

\[
\begin{array}{l}
\text{r\_on} \\
\quad \text{when} \quad r = 0 \\
\quad \quad a = 1 \\
\quad \text{then} \\
\quad \quad r := 1 \\
\quad \quad cr := cr + 1 \\
\quad \text{end}
\end{array}
\]

\[
\begin{align*}
a &= 1 \land r = 1 & \Rightarrow & & cr = ca \\
r &= 0 \\
a &= 1 \\
\vdash \\
\begin{align*}
a &= 1 \land 1 = 1 & \Rightarrow & & cr + 1 = ca
\end{align*}
\]

We have forgotten invariants $\text{pat1}_1$ and $\text{pat0}_6$
What about Event r_on and Invariant pat1_3

- Invariant \text{pat1}_3

\[ a = 1 \land r = 1 \Rightarrow cr = ca \]

- Event r_on

\begin{align*}
\text{r_on} \\
\quad \text{when} r = 0 \\
\quad a = 1 \\
\quad \text{then} \quad r &:= 1 \\
\quad cr &:= cr + 1 \\
\quad \text{end}
\end{align*}

\begin{align*}
\quad r &= 0 \land a = 1 \Rightarrow cr < ca \\
\quad ca &\leq cr + 1 \\
\quad a &= 1 \land r = 1 \Rightarrow cr = ca \\
\quad r &= 0 \\
\quad a &= 1 \\
\quad \vdash a &= 1 \land 1 = 1 \Rightarrow cr + 1 = ca
\end{align*}

- Everything is proved now

- We do not need to add more invariants
Summary of the Events for the **Strong Reaction**

The counters have been removed.

**Initial Conditions:**

\[
\begin{align*}
\text{init} & : \quad a &:= 0, \\
& \quad r &:= 0
\end{align*}
\]

**a_on**

\[
\begin{align*}
\text{when} & : \quad a &= 0, \\
& \quad r &= 0, \\
\text{then} & : \quad a &:= 1, \\
\text{end} &
\end{align*}
\]

**a_off**

\[
\begin{align*}
\text{when} & : \quad a &= 1, \\
& \quad r &= 1, \\
\text{then} & : \quad a &:= 0, \\
\text{end} &
\end{align*}
\]

**r_on**

\[
\begin{align*}
\text{when} & : \quad r &= 0, \\
& \quad a &= 1, \\
\text{then} & : \quad r &:= 1, \\
\text{end} &
\end{align*}
\]

**r_off**

\[
\begin{align*}
\text{when} & : \quad r &= 1, \\
& \quad a &= 0, \\
\text{then} & : \quad r &:= 0, \\
\text{end} &
\end{align*}
\]
Strong Synchronization of Events

\[
\begin{array}{c}
a_{\text{on}} \\
\downarrow \\
r_{\text{on}} \\
\downarrow \\
a_{\text{off}} \\
\downarrow \\
r_{\text{off}} \\
\downarrow \\
a_{\text{on}} \\
\end{array}
\]
- Putting together these two invariants

\[
\text{pat1.2:} \quad a = 0 \implies ca = cr \\
\text{pat1.3:} \quad a = 1 \land r = 1 \implies cr = ca
\]

- leads to the following

\[
\text{pat1.4:} \quad a = 0 \lor r = 1 \implies ca = cr
\]
Simplifying the Invariants

\[
\begin{align*}
\text{pat0.5:} & \quad cr \leq ca \\
\text{pat0.6:} & \quad a = 1 \land r = 0 \Rightarrow cr < ca \\
\text{pat1.1:} & \quad ca \leq cr + 1 \\
\text{pat1.4:} & \quad a = 0 \lor r = 1 \Rightarrow ca = cr
\end{align*}
\]

This can be simplified to

\[
\begin{align*}
\text{pat2.1:} & \quad a = 1 \land r = 0 \Rightarrow ca = cr + 1 \\
\text{pat2.2:} & \quad a = 0 \lor r = 1 \Rightarrow ca = cr
\end{align*}
\]
pat0_1: \( a \in \{0, 1\} \)

pat0_2: \( r \in \{0, 1\} \)

pat0_3: \( ca \in \mathbb{N} \)

pat0_4: \( cr \in \mathbb{N} \)

pat2_1: \( a = 1 \land r = 0 \Rightarrow ca = cr + 1 \)

pat2_2: \( a = 0 \lor r = 1 \Rightarrow ca = cr \)
Comparing the two Invariants

**pat2_1**: \( a = 1 \land r = 0 \Rightarrow ca = cr + 1 \)

**pat2_2**: \( a = 0 \lor r = 1 \Rightarrow ca = cr \)

- \( a = 0 \land r = 0 \) \( \Rightarrow \) \( ca = cr \)
- \( a = 1 \land r = 0 \) \( \Rightarrow \) \( ca = cr + 1 \)
- \( a = 0 \lor r = 1 \) \( \Rightarrow \) \( ca = cr \)
- \( a = 1 \lor r = 1 \) \( \Rightarrow \) \( ca = cr + 1 \)
What we Have Learned

- Proof failures helped us improving our models

- When an invariant preservation proof fails on an event, there are two solutions:
  - adding a new invariant
  - strengthening the guard

- Modelling considerations helped us choosing one or the other

- At the end, we reached a stable situation (fixpoint)
3. Writing the Requirement Document
<table>
<thead>
<tr>
<th>Equipment Description</th>
<th>EQP_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>The system has got the following pieces of equipment: a Motor, a Clutch, and a Door</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Button Description</th>
<th>EQP_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four Buttons are used to start and stop the motor, and engage and disengage the clutch</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controller Description</th>
<th>EQP_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Controller is supposed to manage this equipment</td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Buttons and Controller are weakly synchronized</td>
<td>FUN_1</td>
</tr>
<tr>
<td>Controller are Equipment are strongly synchronized</td>
<td>FUN_2</td>
</tr>
<tr>
<td>Requirement</td>
<td>SAF</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>When the clutch is engaged, the motor must work</td>
<td>SAF_1</td>
</tr>
<tr>
<td>When the clutch is engaged, the door must be closed</td>
<td>SAF_2</td>
</tr>
</tbody>
</table>
### Requirements: Relationship Between Door and Clutch

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE</td>
<td>FUN_3</td>
</tr>
<tr>
<td>When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE</td>
<td>FUN_4</td>
</tr>
<tr>
<td>Opening and closing the door are not independent. It must be synchronized with disengaging and engaging the clutch</td>
<td>FUN_5</td>
</tr>
</tbody>
</table>
Overview
4. Proposing a Refinement Strategy
- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor buttons to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd refinement: Constraining the clutch and the motor
- 4th refinement: Connecting the controller to the door

- 5th refinement: Constraining the clutch and the door

- 6th refinement: More constraints between clutch and door

- 7th refinement: Connecting the clutch buttons to the controller
5. Development of the Model using Refinements and Design Patterns
Initial Model: Connecting the Controller to the Motor

Controller are Equipment are strongly synchronized

Controller

Motor

Strong Reaction

FUN_2
Model for **strong** action and reaction: the Final Events

The counters have been removed.

**init**

\[
\begin{align*}
a & := 0 \\
r & := 0
\end{align*}
\]

**a_on**

\[
\begin{align*}
\text{when} & \quad a = 0 \\
\quad & \quad r = 0 \\
\text{then} & \quad a := 1 \\
\text{end}
\end{align*}
\]

**a_off**

\[
\begin{align*}
\text{when} & \quad a = 1 \\
\quad & \quad r = 1 \\
\text{then} & \quad a := 0 \\
\text{end}
\end{align*}
\]

**r_on**

\[
\begin{align*}
\text{when} & \quad r = 0 \\
\quad & \quad a = 1 \\
\text{then} & \quad r := 1 \\
\text{end}
\end{align*}
\]

**r_off**

\[
\begin{align*}
\text{when} & \quad r = 1 \\
\quad & \quad a = 0 \\
\text{then} & \quad r := 0 \\
\text{end}
\end{align*}
\]
**Initial Model: the Context**

- **set:** $STATUS$
- **constants:** 
  - stopped
  - working

**axm0.1:** $STATUS = \{stopped, working\}$

**axm0.2:** stopped $\neq$ working
Initial Model: the State

variables:  

\[ \begin{align*} 
\text{motor}_\text{actuator} \\
\text{motor}_\text{sensor} 
\end{align*} \]

inv0_1:  \[ \text{motor}_\text{sensor} \in STATUS \]

inv0_2:  \[ \text{motor}_\text{actuator} \in STATUS \]
Initial Model: the Synchronization

Controller

Action

MotorController

motor_actuator

Strong Reaction

motor_sensor

Motor

Strong Reaction
- We instantiate the weak pattern as follows:

\[
\begin{align*}
a & \rightsquigarrow motor\_actuator \\
r & \rightsquigarrow motor\_sensor \\
0 & \rightsquigarrow stopped \\
1 & \rightsquigarrow working \\
a\_on & \rightsquigarrow treat\_start\_motor \\
a\_off & \rightsquigarrow treat\_stop\_motor \\
r\_on & \rightsquigarrow Motor\_start \\
r\_off & \rightsquigarrow Motor\_stop \\
\end{align*}
\]

- Convention: Controller events start with "treat_"
Initial Model: Initialization

\[
\begin{align*}
\text{init} \\
& a := 0 \\
& r := 0
\end{align*}
\]

\[
\begin{align*}
\text{init} \\
& \text{motor\_actuator} := \text{stopped} \\
& \text{motor\_sensor} := \text{stopped}
\end{align*}
\]
a_on

when

\[ a = 0 \]
\[ r = 0 \]

then

\[ a := 1 \]

end

treat_start_motor

when

\[ \text{motor}_{\text{actuator}} = \text{stopped} \]
\[ \text{motor}_{\text{sensor}} = \text{stopped} \]

then

\[ \text{motor}_{\text{actuator}} := \text{working} \]

end
Initial Model: Environment Event (1)

\[
\begin{align*}
\text{r} & \text{on} \\
\text{when} & r = 0 \\
\text{ then } & a = 1 \\
\text{ end} & r := 1
\end{align*}
\]

\[
\begin{align*}
\text{Motor \_start} \\
\text{when} & \text{motor \_sensor = stopped} \\
\text{ then } & \text{motor \_actuator = working} \\
\text{ then } & \text{motor \_sensor := working} \\
\text{ end}
\end{align*}
\]
Initial Model: Controller Events (2)

a_off
when
  $a = 1$
  $r = 1$
then
  $a := 0$
end

treat_stop_motor
when
  motor_actuator = working
  motor_sensor = working
then
  motor_actuator := stopped
end
**Initial Model: Environment Event (2)**

- **r_off**
  - **when**
    - $r = 1$
    - $a = 0$
  - **then**
    - $r := 0$
  - **end**

- **Motor_stop**
  - **when**
    - $motor\_sensor = working$
    - $motor\_actuator = stopped$
  - **then**
    - $motor\_sensor := stopped$
  - **end**
Initial Model: Summary of the Events

- Environment
  - motor_start
  - motor_stop

- Controller
  - treat_start_motor
  - treat_stop_motor
Buttons and Controller are weakly synchronized

<table>
<thead>
<tr>
<th></th>
<th>FUN_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak Reaction</td>
<td></td>
</tr>
<tr>
<td>Strong Reaction</td>
<td></td>
</tr>
</tbody>
</table>
The counters have been removed.

\[
\text{a}_{\text{on}} \quad \text{when} \quad a = 0 \\
\text{then} \quad a := 1 \\
\text{end}
\]

\[
\text{a}_{\text{off}} \quad \text{when} \quad a = 1 \\
\text{then} \quad a := 0 \\
\text{end}
\]

\[
\text{r}_{\text{on}} \quad \text{when} \quad r = 0 \\
\text{then} \quad a := 1 \\
\text{then} \quad r := 1 \\
\text{end}
\]

\[
\text{r}_{\text{off}} \quad \text{when} \quad r = 1 \\
\text{then} \quad a := 0 \\
\text{then} \quad r := 0 \\
\text{end}
\]
First Refinement: the State

variables: ...  

\textit{start\_motor\_button}  
\textit{stop\_motor\_button}  
\textit{start\_motor\_impulse}  
\textit{stop\_motor\_impulse}  

\begin{align*}  
\text{inv1\_1:} & \quad \textit{stop\_motor\_button} \in \text{BOOL} \\
\text{inv1\_2:} & \quad \textit{start\_motor\_button} \in \text{BOOL} \\
\text{inv1\_3:} & \quad \textit{stop\_motor\_impulse} \in \text{BOOL} \\
\text{inv1\_4:} & \quad \textit{start\_motor\_impulse} \in \text{BOOL} 
\end{align*}
First Refinement: the State

CONTROLLER

start_motor_button

start_motor_impulse

weak reaction

stop_motor_impulse

Stop Button

action

stop_motor_button

action

Start Button

weak reaction
- We instantiate the pattern as follows:

\[
a \mapsto start\_motor\_button
\]
\[
r \mapsto start\_motor\_impulse
\]
\[
0 \mapsto FALSE
\]
\[
1 \mapsto TRUE
\]

- We rename `treat_start_motor` as `treat_push_start_motor_button`
1st Refinement: Refinement of Initialization

init

\[ a := 0 \]
\[ r := 0 \]

\begin{align*}
\text{init} \\
\quad & \text{motor}_{-}\text{actuator} := \text{stopped} \\
\quad & \text{motor}_{-}\text{sensor} := \text{stopped} \\
\quad & \text{start}_{-}\text{motor}_{-}\text{button} := \text{FALSE} \\
\quad & \text{start}_{-}\text{motor}_{-}\text{impulse} := \text{FALSE}
\end{align*}
a_on
  when
    a = 0
  then
    a := 1
end

push_start_motor_button
  when
    start_motor_button = FALSE
  then
    start_motor_button := TRUE
end

a_off
  when
    a = 1
  then
    a := 0
end

release_start_motor_button
  when
    start_motor_button = TRUE
  then
    start_motor_button := FALSE
end
- This is the most important slide of the talk
- We can see how patterns can be superposed
a_on
when
  a = 0
  r = 0
then
  a := 1
end

treat_start_motor
when
  motor_actuator = stopped
  motor_sensor = stopped
then
  motor_actuator := working
end

treat_push_start_motor_button
when
  start_motor_impulse = FALSE
  start_motor_button = TRUE
  motor_actuator = stopped
  motor_sensor = stopped
then
  start_motor_impulse := TRUE
  motor_actuator := working
end

r_on
when
  r = 0
  a = 1
then
  r := 1
end
Design Pattern Integration within a Development

Initial Model

refines

Refinement 1

refines

Refinement n

refines

Instantiated Pattern
r_off
  when
    r = 1
    a = 0
  then
    r := 0
  end

\[
\text{treat\_release\_start\_motor\_button}
\]

\[
\text{when}
\]

\[
\begin{align*}
&\text{start\_motor\_impulse} = \text{TRUE} \\
&\text{start\_motor\_button} = \text{FALSE}
\end{align*}
\]

\[
\text{then}
\]

\[
\begin{align*}
&\text{start\_motor\_impulse} := \text{FALSE}
\end{align*}
\]

end
- We instantiate the pattern as follows:

\[
\begin{align*}
a & \mapsto \text{stop\_motor\_button} \\
r & \mapsto \text{stop\_motor\_impulse} \\
0 & \mapsto \text{FALSE} \\
1 & \mapsto \text{TRUE} \\
a\_on & \mapsto \text{push\_stop\_motor\_button} \\
a\_off & \mapsto \text{release\_stop\_motor\_button} \\
r\_on & \mapsto \text{treat\_push\_stop\_motor\_button} \\
r\_off & \mapsto \text{treat\_release\_stop\_motor\_button}
\end{align*}
\]
<table>
<thead>
<tr>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a := 0 )</td>
</tr>
<tr>
<td>( r := 0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{motor}_\text{actuator} := \text{stopped} )</td>
</tr>
<tr>
<td>( \text{motor}_\text{sensor} := \text{stopped} )</td>
</tr>
<tr>
<td>( \text{start}_\text{motor}_\text{button} := \text{FALSE} )</td>
</tr>
<tr>
<td>( \text{start}_\text{motor}_\text{impulse} := \text{FALSE} )</td>
</tr>
<tr>
<td>( \text{stop}_\text{motor}_\text{button} := \text{FALSE} )</td>
</tr>
<tr>
<td>( \text{stop}_\text{motor}_\text{impulse} := \text{FALSE} )</td>
</tr>
</tbody>
</table>
**First Refinement: New Environment Events**

```
\begin{align*}
\text{a\_on} \\
\quad & \text{when} \quad a = 0 \\
\quad & \text{then} \\
\quad & \quad a := 1 \\
\quad & \text{end} \\
\text{push\_stop\_motor\_button} \\
\quad & \text{when} \\
\quad & \quad stop\_motor\_button = \text{FALSE} \\
\quad & \text{then} \\
\quad & \quad stop\_motor\_button := \text{TRUE} \\
\quad & \text{end} \\
\text{a\_off} \\
\quad & \text{when} \quad a = 1 \\
\quad & \text{then} \\
\quad & \quad a := 0 \\
\quad & \text{end} \\
\text{release\_stop\_motor\_button} \\
\quad & \text{when} \\
\quad & \quad stop\_motor\_button = \text{TRUE} \\
\quad & \text{then} \\
\quad & \quad stop\_motor\_button := \text{FALSE} \\
\quad & \text{end}
\end{align*}
```
r_on

when
  \( r = 0 \)
  \( a = 1 \)
then
  \( r := 1 \)
end

treat\_push\_stop\_motor\_button
refines
  treat\_stop\_motor
when
  when
  \( \text{stop\_motor\_impulse} = \text{FALSE} \)
  \( \text{stop\_motor\_button} = \text{TRUE} \)
  \( \text{motor\_sensor} = \text{working} \)
  \( \text{motor\_actuator} = \text{working} \)
then
  \( \text{stop\_motor\_impulse} := \text{TRUE} \)
  \( \text{motor\_actuator} := \text{stopped} \)
end
r_off
  when
      r = 1
      a = 0
  then
      r := 0
  end

\begin{tabular}{|l|}
\hline
\textbf{treat\_release\_stop\_motor\_button} \\
\hline
\textbf{when} \\
\hline
\quad \textit{stop\_motor\_impulse} = \text{TRUE} \\
\quad \textit{stop\_motor\_button} = \text{FALSE} \\
\hline
\textbf{then} \\
\hline
\quad \textit{stop\_motor\_impulse} := \text{FALSE} \\
\hline
\textbf{end} \\
\hline
\end{tabular}
Independent Synchronizations

push_start_motor_button → release_start_motor_button

| treat_push_start_motor_button | treat_release_start_motor_button |
Independent Synchronizations

- push_start_motor_button
- release_start_motor_button
- treat_push_start_motor_button
- treat_release_start_motor_button
- treat_push_stop_motor_button
- treat_release_stop_motor_button
- push_stop_motor_button
- release_stop_motor_button
Independent Synchronizations

`push_start_motor_button` ↔ `release_start_motor_button`

`treat_push_start_motor_button` ↔ `treat_release_start_motor_button`

`treat_release_stop_motor_button` ↔ `treat_push_stop_motor_button`

`release_stop_motor_button` ↔ `push_stop_motor_button`

`treat_push_start_motor_button` ↔ `treat_push_stop_motor_button`

`Motor_start` ↔ `Motor_stop`
Weak and Strong Reactions Together

Start Button  Stop Button

start_motor_button  stop_motor_button

start_motor_impulse  stop_motor_impulse

CONTROLLER

MOTOR

motor_actuator

motor_sensor

weak reaction  weak reaction

strong reaction
Combined Synchronizations

push_start_motor_button

release_start_motor_button

treat_push_start_motor_button

treat_release_start_motor_button

Motor_start

Motor_stop

treat_release_stop_motor_button

treat_push_stop_motor_button

release_stop_motor_button

push_stop_motor_button
Problems with `treat_push_start_motor_button`

```
treat_push_start_motor_button
  refines
treat_start_motor
  when
    start_motor_impulse = FALSE
    start_motor_button = TRUE
    motor_actuator = stopped
    motor_sensor = stopped
  then
    start_motor_impulse := TRUE
    motor_actuator := working
end
```

- What happens when the following hold

  \[ \neg (motor_actuator = \text{stopped} \land motor_sensor = \text{stopped}) \]

- We need another event
- In the second case, the button has been pushed but the internal conditions are not met 
- However, we need to record that the button has been pushed:

\[ \text{start\_motor\_impulse} := \text{TRUE} \]
Problems with \texttt{treat\_push\_stop\_motor\_button}

- In the second case, \textit{the button has been pushed} but the internal conditions are not met.
- However, we need to record that the button has been pushed:

\begin{verbatim}
stop\_motor\_impulse := TRUE
end
\end{verbatim}

\begin{verbatim}
treat\_push\_stop\_motor\_button \textbf{false}
when
\begin{align*}
\text{stop\_motor\_impulse} & = \text{FALSE} \\
\text{stop\_motor\_button} & = \text{TRUE} \\
\text{motor\_sensor} & = \text{working} \\
\text{motor\_actuator} & = \text{working}
\end{align*}
then
\begin{align*}
\text{stop\_motor\_impulse} & := \text{TRUE} \\
\text{motor\_actuator} & := \text{stopped}
\end{align*}
end
\end{verbatim}
- Environment
  - motor_start
  - motor_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
2nd Refinement: Connecting the Controller to the Clutch

Start Button

Stop Button

start_motor_button

stop_motor_button

clutch_actuator

clutch_sensor

motor_actuator

motor_sensor

CONTROLLER
- We introduce the set in a new context:

\[ CLUTCH = \{\text{engaged}, \text{disengaged}\} \]

- We copy the initial model where we instantiate:

\[
\begin{align*}
\text{motor} & \leadsto \text{clutch} \\
\text{STATUS} & \leadsto CLUTCH \\
\text{working} & \leadsto \text{engaged} \\
\text{stopped} & \leadsto \text{disengaged}
\end{align*}
\]
- Environment
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
- An additional **safety constraint**

| When the clutch is engaged, the motor must work | SAF_1 |

- For this we develop **ANOTHER DESIGN PATTERN**

- It is called: **Weak synchronization of two Strong Reactions**
When the clutch is engaged

then

the motor must work
When the clutch is engaged
then
the motor must work

Pattern: Weak Synchronization of Strong Reactions

\[ s = 1 \Rightarrow r = 1 \]
The Synchronization is **Weak** (1)

When the clutch is disengaged, then the motor can be started and stopped several times.
The Synchronization is **Weak** (2)

When the motor works, then the clutch can be engaged and disengaged several times.
Putting the Two Together
Synchronizing the Reactions Without Touching them

\[ \text{a}_{\text{on}} \quad \text{a}_{\text{off}} \]
\[ \text{r}_{\text{on}} \quad \text{r}_{\text{off}} \]
\[ \text{b}_{\text{on}} \quad \text{b}_{\text{off}} \]
\[ \text{s}_{\text{on}} \quad \text{s}_{\text{off}} \]
The Initial State Situation

dbl0_1: \( a \in \{0, 1\} \)
dbl0_2: \( r \in \{0, 1\} \)
dbl0_3: \( ca \in \mathbb{N} \)
dbl0_4: \( cr \in \mathbb{N} \)
dbl0_5: \( a = 1 \land r = 0 \Rightarrow ca = cr + 1 \)
dbl0_6: \( a = 0 \lor r = 1 \Rightarrow ca = cr \)

dbl0_7: \( b \in \{0, 1\} \)
dbl0_8: \( s \in \{0, 1\} \)
dbl0_9: \( cb \in \mathbb{N} \)
dbl0_10: \( cs \in \mathbb{N} \)
dbl0_11: \( b = 1 \land s = 0 \Rightarrow cb = cs + 1 \)
dbl0_12: \( b = 0 \lor s = 1 \Rightarrow cb = cs \)
The Initial Event Situation (1)

\[
\begin{align*}
\text{a_on} & \\
\text{when} & \\
\quad a &= 0 \\
\quad r &= 0 \\
\text{then} & \\
\quad a &= 1 \\
\quad ca &= ca + 1 \\
\text{end} & \\
\text{r_on} & \\
\text{when} & \\
\quad r &= 0 \\
\quad a &= 1 \\
\text{then} & \\
\quad r &= 1 \\
\quad cr &= cr + 1 \\
\text{end} & \\
\text{a_off} & \\
\text{when} & \\
\quad a &= 1 \\
\quad r &= 1 \\
\text{then} & \\
\quad a &= 0 \\
\text{end} & \\
\text{r_off} & \\
\text{when} & \\
\quad r &= 1 \\
\quad a &= 0 \\
\text{then} & \\
\quad r &= 0 \\
\text{end} & 
\end{align*}
\]
The Initial Event Situation (2)

\begin{align*}
\text{b.on} & \\
\text{when} & \\
& b = 0 \\
& s = 0 \\
\text{then} & \\
& b := 1 \\
& cb := cb + 1 \\
\text{end} & \\
\text{s.on} & \\
\text{when} & \\
& s = 0 \\
& b = 1 \\
\text{then} & \\
& s := 1 \\
& cs := cs + 1 \\
\text{end} & \\
\text{b.off} & \\
\text{when} & \\
& b = 1 \\
& s = 1 \\
\text{then} & \\
& b := 0 \\
\text{end} & \\
\text{s.off} & \\
\text{when} & \\
& s = 1 \\
& b = 0 \\
\text{then} & \\
& s := 0 \\
\text{end} &
\end{align*}
The Synchronizing Invariant

\[ \text{dbl1}_1: \quad s = 1 \implies r = 1 \]

- It seems sufficient to add the following guards

\[
\begin{aligned}
\text{s\_on} \\
\text{when} \\
\quad s = 0 \\
\quad b = 1 \\
\quad r = 1 \\
\text{then} \\
\quad s := 1 \\
\quad cs := cs + 1 \\
\text{end}
\end{aligned}
\]

\[
\begin{aligned}
\text{r\_off} \\
\text{when} \\
\quad r = 1 \\
\quad a = 0 \\
\quad s = 0 \\
\text{then} \\
\quad r := 0 \\
\text{end}
\end{aligned}
\]

- But we do not want to touch these events
Introducing Additional Invariants to Remove the red guards

- We introduce the following additional invariants

\[ \text{dbl1\_2: } b = 1 \Rightarrow r = 1 \]

\[ \text{dbl1\_3: } a = 0 \Rightarrow s = 0 \]
Maintaining Invariant dbl1_2 (1)

\[
\text{dbl1}_2: \quad b = 1 \implies r = 1
\]

In order to maintain this invariant, we have to refine \texttt{b\_on}

\[
\begin{aligned}
\text{b\_on} \\
\text{when} \quad b = 0 \\
\quad s = 0 \\
\text{then} \\
\quad b := 1 \\
\quad cb := cb + 1 \\
\text{end}
\end{aligned}
\]

\[
\begin{aligned}
\text{b\_on} \\
\text{when} \quad b = 0 \\
\quad s = 0 \\
\text{then} \\
\quad r = 1 \\
\quad b := 1 \\
\quad cb := cb + 1 \\
\text{end}
\end{aligned}
\]
In order to maintain this invariant, we have to refine `r_off`.

- But, again, we do not want to touch this event.
Introducing a new invariant to Remove the Red Guard

- We introduce the following invariant

\[
\text{dbl1.4: } a = 0 \implies b = 0
\]
Maintaining Invariant dbl1_3 (1)

dbl1_3: \( a = 0 \Rightarrow s = 0 \)

In order to maintain this invariant, we have to refine `a_off`

\[
\begin{array}{l}
a_{\text{off}} \\
\text{when} \\
a = 1 \\
r = 1 \\
\text{then} \\
a := 0 \\
\text{end}
\end{array}
\quad \sim \quad
\begin{array}{l}
a_{\text{off}} \\
\text{when} \\
a = 1 \\
r = 1 \\
s = 0 \\
\text{then} \\
a := 0 \\
\text{end}
\end{array}
\]
Maintaining (Contraposition of) Invariant \texttt{dbl1\_3} (2)

\begin{align*}
\texttt{dbl1\_3: } & \quad a = 0 \implies s = 0 \quad (s = 1 \implies a = 1) \\
\end{align*}

In order to maintain this invariant, we have to refine \texttt{s\_on}

\begin{align*}
\texttt{s\_on} & \\
\texttt{\hspace{1cm} when} & \\
& \quad s = 0 \\
& \quad b = 1 \\
\texttt{\hspace{1cm} then} & \\
& \quad s := 1 \\
& \quad cs := cs + 1 \\
\texttt{end} & \\
\end{align*}

\begin{align*}
\texttt{s\_on} & \\
\texttt{\hspace{1cm} when} & \\
& \quad s = 0 \\
& \quad b = 1 \\
& \quad a = 1 \\
\texttt{\hspace{1cm} then} & \\
& \quad s := 1 \\
& \quad cs := cs + 1 \\
\texttt{end} & \\
\end{align*}

- But, again, we do not want to touch this event
Introducing a new invariant to Remove the Red Guard

\[
\begin{align*}
\text{s\_on} & \\
\text{\textbf{when}} & \\
\quad s &= 0 \\
\quad b &= 1 \\
\quad a &= 1 \\
\text{\textbf{then}} & \\
\quad s &:= 1 \\
\quad cs &:= cs + 1 \\
\text{\textbf{end}} &
\end{align*}
\]

- We have to introduce the following invariant

\[b = 1 \Rightarrow a = 1\]

- Fortunately, this is \texttt{dbl1\_4} \(a = 0 \Rightarrow b = 0\) contraposed
In order to maintain this invariant, we have to refine `a_off` again.

```plaintext
a_off
when
  a = 1
  r = 1
  s = 0
then
  a := 0
end

\[ \sim \]

a_off
when
  a = 1
  r = 1
  s = 0
then
  a := 0
end
```

 dbl1_4: \( a = 0 \implies b = 0 \)
In order to maintain this invariant, we have to refine \texttt{b.on} again

\begin{verbatim}
    b_on
    when
        b = 0
        s = 0
        r = 1
    then
        b, cb := 1, cb + 1
    end

\end{verbatim}
Summary of Refinement: Reactions have not been Touched

\[
dbl1_1: \quad s = 1 \implies r = 1 \\
dbl1_2: \quad b = 1 \implies r = 1 \\
dbl1_3: \quad a = 0 \implies s = 0 \\
dbl1_4: \quad a = 0 \implies b = 0
\]

b_on
\[
\begin{align*}
\text{when} & \quad b = 0 \\
& \quad s = 0 \\
& \quad r = 1 \\
& \quad a = 1 \\
\text{then} & \quad b, cb := 1, cb + 1 \\
\end{align*}
\]

end

a_off
\[
\begin{align*}
\text{when} & \quad a = 1 \\
& \quad r = 1 \\
& \quad s = 0 \\
& \quad b = 0 \\
\text{then} & \quad a := 0 \\
\end{align*}
\]

end
Intuition about the Invariants

\[
\begin{align*}
\text{dbl1}_1: & \quad s = 1 \Rightarrow r = 1 \\
\text{dbl1}_2: & \quad b = 1 \Rightarrow r = 1 \\
\text{dbl1}_3: & \quad a = 0 \Rightarrow s = 0 \quad (s = 1 \Rightarrow a = 1) \\
\text{dbl1}_4: & \quad a = 0 \Rightarrow b = 0 \quad (b = 1 \Rightarrow a = 1)
\end{align*}
\]

This can be put into a single invariant

\[
\text{dbl1}_5: \quad b = 1 \lor s = 1 \Rightarrow a = 1 \land r = 1
\]

with the following contraposited form

\[
\text{dbl1}_6: \quad a = 0 \lor r = 0 \Rightarrow b = 0 \land s = 0
\]
Intuition about the Invariants

Reminder: --- is the **motor** and --- is the **clutch**

**dbl1_5:** \( b = 1 \lor s = 1 \Rightarrow a = 1 \land r = 1 \)

**dbl1_6:** \( a = 0 \lor r = 0 \Rightarrow b = 0 \land s = 0 \)
Weak Synchronization of Strong Reaction: the Problem

Diagram showing the interactions between variables a_on, a_off, r_on, r_off, b_on, b_off, s_on, s_off.
Weak Synchronization of Strong Reaction: the Solution
When the clutch is engaged, the motor must work

\[ \text{inv3}_1: \quad \text{clutch\_sensor} = \text{engaged} \]
\[ \Rightarrow \]
\[ \text{motor\_sensor} = \text{working} \]

- This is an instance of the previous design pattern
- We instantiate the pattern as follows:

\[
\begin{align*}
  a & \leadsto \text{motor} \_ \text{actuator} & a \_ \text{on} & \leadsto & \text{treat} \_ \text{push} \_ \text{start} \_ \text{motor} \_ \text{button} \\
  r & \leadsto \text{motor} \_ \text{sensor} & a \_ \text{off} & \leadsto & \text{treat} \_ \text{push} \_ \text{stop} \_ \text{motor} \_ \text{button} \\
  0 & \leadsto \text{stopped} & r \_ \text{on} & \leadsto & \text{Motor} \_ \text{start} \\
  1 & \leadsto \text{working} & r \_ \text{off} & \leadsto & \text{Motor} \_ \text{stop} \\
  b & \leadsto \text{clutch} \_ \text{actuator} & b \_ \text{on} & \leadsto & \text{treat} \_ \text{start} \_ \text{clutch} \\
  s & \leadsto \text{clutch} \_ \text{sensor} & b \_ \text{off} & \leadsto & \text{treat} \_ \text{stop} \_ \text{clutch} \\
  0 & \leadsto \text{disengaged} & s \_ \text{on} & \leadsto & \text{Clutch} \_ \text{start} \\
  1 & \leadsto \text{engaged} & s \_ \text{off} & \leadsto & \text{Clutch} \_ \text{stop}
\end{align*}
\]
Translating the pattern invariants (1)

<table>
<thead>
<tr>
<th>db111: $s = 1 \Rightarrow r = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>db112: $b = 1 \Rightarrow r = 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv31: $clutch_sensor = engaged$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$motor_sensor = working$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv32: $clutch_actuator = engaged$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$motor_sensor = working$</td>
</tr>
</tbody>
</table>
Translating the pattern invariants (2)

<table>
<thead>
<tr>
<th>dbl1_3:</th>
<th>$a = 0 \implies s = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbl1_4:</td>
<td>$a = 0 \implies b = 0$</td>
</tr>
</tbody>
</table>

| inv3_3: | $motor\_actuator = stopped$  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$clutch_sensor = disengaged$</td>
</tr>
</tbody>
</table>
| inv3_4: | $motor\_actuator = stopped$  
|         | $clutch\_actuator = disengaged$ |
Adapting the Events of the Pattern (1)

```plaintext
b_on
when
    b = 0
    s = 0
    r = 1
    a = 1
then
    b := 1
end

end

treat_start_clutch
    when
        clutch_actuator = disengaged
        clutch_sensor = disengaged
        motor_sensor = working
        motor_actuator = working
    then
        clutch_actuator := engaged
    end
```

**Note:** The above text describes a pattern for adapting events, specifically focusing on conditions for activating or deactivating a clutch based on sensor readings. The pattern is implemented in a block `treat_start_clutch` with conditions for activation and deactivation based on sensor states.
Adapting the events of the pattern (2)

a_off
when
  a = 1
  r = 1
  s = 0
  b = 0
then
  a := 0
end

treat_push_stop_motor_button
  when
    stop_motor_impulse = FALSE
    stop_motor_button = TRUE
    motor_actuator = working
    motor_sensor = working
    clutch_sensor = disengaged
    clutch_actuator = disengaged
  then
    motor_actuator := stopped
    stop_motor_impulse := TRUE
end
- Environment (no new events)
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
Fourth Refinement: Connecting the Controller to the Door

- We copy (after renaming "motor" to "door") what has been done in the initial model
- We introduce the set in a new context:

\[ DOOR = \{\text{open}, \text{closed}\} \]

- We copy the initial model where we instantiate:

\[
\begin{align*}
\text{motor} & \sim \text{door} \\
\text{STATUS} & \sim DOOR \\
\text{working} & \sim \text{closed} \\
\text{stopped} & \sim \text{open}
\end{align*}
\]
- Environment
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - door_close
  - door_open
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door
- An additional safety constraint

| When the **clutch** is engaged, the **door** must be closed | SAF_2 |

- We copy (after renaming "motor" to "door") what has been done in the third model:

| When the **clutch** is engaged, the **motor** must work | SAF_1 |
Fifth Reft.: Something was forgotten Concerning the Door
- Can you guess it?
- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated
- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

| When the motor is stopped, the door must be open | SAF_3 |
- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

<table>
<thead>
<tr>
<th>When the door is closed, the motor must work</th>
<th>SAF_3’</th>
</tr>
</thead>
</table>

- SAF_3’ is the contraposed form of SAF_3
- Additional safety constraint

<table>
<thead>
<tr>
<th>When the door is closed, the motor must work</th>
<th>SAF_3’</th>
</tr>
</thead>
</table>

- We copy (after renaming "clutch" to "door") what has been done in the third model:

| When the clutch is engaged, the motor must work | SAF_1  |
When the **clutch** is engaged, the **motor** must work  

<table>
<thead>
<tr>
<th>Requirement SAF_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the <strong>clutch</strong> is engaged, the <strong>door</strong> must be closed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement SAF_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the <strong>clutch</strong> is engaged, the <strong>door</strong> must be closed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement SAF_3'</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the <strong>door</strong> is closed, the <strong>motor</strong> must work</td>
</tr>
</tbody>
</table>

- Requirement SAF_1 is now redundant: \( SAF_2 \wedge SAF_3' \Rightarrow SAF_1 \)
Possible New Refinement Strategy

- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor button to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd (4th) refinement: Connecting the controller to the door
- 4th (5th) refinement: Constraining the clutch and the door
  Constraining the motor and the door

- 5th (6th) refinement: More constraints between clutch and door

- 6th (7th) refinement: Connecting the clutch button to the controller
- Environment (no new events)
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - door_close
  - door_open
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door
- Adding two functional constraints

<table>
<thead>
<tr>
<th>When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE</th>
<th>FUN_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE</td>
<td>FUN_4</td>
</tr>
</tbody>
</table>
- When the clutch is disengaged, the door cannot be closed several times.
Problem with the **Weak Synchronization of Strong Reactions**

- When the door is closed, the clutch cannot be disengaged several times.
What we want:

\[
ca = cb \lor ca = cb + 1
\]

\[
cr = cs \lor cr = cs + 1
\]
How about counters $ca$ and $cb$?

$ca = cb$

$ca = cb + 1$
In Search of a Solution

ca = cb

ca = cb + 1

a = 1

and

b = 0

b = 0

a = 1 and b = 0

ca = cb

ca = cb + 1
In Search of a Solution

\[ a = 1 \land b = 0 \Rightarrow ca = cb + 1 \]

\[ a = 1 \]

\[ b = 0 \]
This Solution Does not Work

\[ a = 1 \] and \[ b = 0 \]

\[ ca = cb + 1 \]

\[ a = 1 \] and \[ b = 0 \]
The Solution: an Additional Variable $m$

$m = 1 \Rightarrow ca = cb + 1$

$m = 0 \Rightarrow ca = cb$
The Events

m = 0

m = 1

a_on

b_on

a_off
The Modified Events

\begin{align*}
\text{a_on} & \\
\text{when} & \\
& a = 0 \\
& r = 0 \\
\text{then} & \\
& a := 1 \\
& ca := ca + 1 \\
& m := 1 \\
\text{end} & \\
\text{b_on} & \\
\text{when} & \\
& r = 1 \\
& a = 1 \\
& b = 0 \\
& s = 0 \\
& m = 1 \\
\text{then} & \\
& b := 1 \\
& cb := cb + 1 \\
& m := 1 \\
\text{end} & \\
\text{a_off} & \\
\text{when} & \\
& a = 1 \\
& r = 1 \\
& b = 0 \\
& s = 0 \\
& m = 0 \\
\text{then} & \\
& a := 0 \\
\text{end} &
\end{align*}
How about counters $cr$ and $cs$?
In Search of a Solution

m = 0

m = 1
cr = cs
cr = cs+1

r = 1 and s = 0

s = 0

r = 1

s = 0

m = 1

m = 0

m = 0
In Search of a Solution

\[ r = 1 \land s = 0 \implies cr = cs + 1 \]
This Solution Does not Work

\[ m = 0 \]

\[ m = 1 \]

\[ cr = c_{sr} = c_s + 1 \]

\[ cr = c_s \]

\[ r = 1 \] and \[ s = 0 \]

\[ r = 1 \] and \[ s = 0 \]
The Solution

\[ r = 1 \text{ and } s = 0 \]

\[ m = 1 \]

\[ b = 1 \]

\[ m = 0 \]

\[ s = 0 \]

\[ b = 0 \]

\[ m = 0 \]
The Solution

\[ r = 1 \land s = 0 \land (m = 1 \lor b = 1) \Rightarrow cr = cs + 1 \]

\[ r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \Rightarrow cr = cs \]
Summary of Refinement: the state

\[
\begin{align*}
\text{dbl2}_1: & \quad m \in \{0, 1\} \\
\text{dbl2}_2: & \quad m = 1 \implies ca = cb + 1 \\
\text{dbl2}_3: & \quad m = 0 \implies ca = cb \\
\text{dbl2}_4: & \quad r = 1 \land s = 0 \land (m = 1 \lor b = 1) \implies cr = cs + 1 \\
\text{dbl2}_5: & \quad r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \implies cr = cs
\end{align*}
\]
**Summary of Refinement: the state**

\[
\begin{align*}
\text{dbl2.1: } & \quad m \in \{0, 1\} \\
\text{dbl2.2: } & \quad m = 1 \implies ca = cb + 1 \\
\text{dbl2.3: } & \quad m = 0 \implies ca = cb \\
\text{dbl2.4: } & \quad r = 1 \land s = 0 \land (m = 1 \lor b = 1) \implies cr = cs + 1 \\
\text{dbl2.5: } & \quad r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \implies cr = cs
\end{align*}
\]

- The following theorems are easy to prove

\[
\begin{align*}
\text{thm2.1: } & \quad ca = cb \lor ca = cb + 1 \\
\text{thm2.2: } & \quad cr = cs \lor cr = cs + 1
\end{align*}
\]
More Invariants

dbl2.1: \( m \in \{0, 1\} \)

dbl2.2: \( m = 1 \implies ca = cb + 1 \)

dbl2.3: \( m = 0 \implies ca = cb \)

dbl2.4: \( r = 1 \land s = 0 \land (m = 1 \lor b = 1) \implies cr = cs + 1 \)

dbl2.5: \( r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \implies cr = cs \)

dbl2.6: \( r = 1 \land a = 0 \implies m = 0 \)

dbl2.7: \( m = 1 \implies s = 0 \)

- The two new invariants were discovered while doing the proof

- The proofs are now completely automatic
\[ m = 1 \]
\[ r = 1 \]
\[ a = 0 \]
\[ m = 0 \]
\[ s = 0 \]
Instantiation

door closed

\text{treat\_close\_door} \quad \text{(a\_on)} \quad \text{treat\_open\_door} \quad \text{(a\_off)}

treat\_start\_clutch \quad \text{(b\_on)}

clutch engaged
- We instantiate the pattern as follows:

\[
\begin{align*}
a & \mapsto \text{door\_actuator} & b & \mapsto \text{clutch\_actuator} \\
r & \mapsto \text{door\_sensor} & s & \mapsto \text{clutch\_sensor} \\
0 & \mapsto \text{open} & 0 & \mapsto \text{disengaged} \\
1 & \mapsto \text{closed} & 1 & \mapsto \text{engaged} \\
\end{align*}
\]

\[
\begin{align*}
a\_on & \mapsto \text{treat\_close\_door} \\
a\_off & \mapsto \text{treat\_open\_door} \\
b\_on & \mapsto \text{treat\_start\_clutch}
\end{align*}
\]
6th Refinement: Adapting the events of the pattern (2)

\begin{align*}
\text{a_on} & \\
\text{when} & \\
\quad a &= 0 \\
\quad r &= 0 \\
\text{then} & \\
\quad a &:= 1 \\
\quad m &:= 1 \\
\text{end} & \\
\text{treat_close_door} & \\
\text{when} & \\
\quad \text{door_actuator} &= \text{open} \\
\quad \text{door_sensor} &= \text{open} \\
\quad \text{motor_actuator} &= \text{working} \\
\quad \text{motor_sensor} &= \text{working} \\
\text{then} & \\
\quad \text{door_actuator} &:= \text{closed} \\
\quad m &:= 1 \\
\text{end} &
\end{align*}
\begin{align*}
b_{on}
\quad \text{when} \\
\quad b = 0 \\
\quad s = 0 \\
\quad r = 1 \\
\quad a = 1 \\
\quad m = 1
\quad \text{then} \\
\quad b := 1 \\
\quad m := 0 \\
\quad \text{end}
\end{align*}

\begin{align*}
\text{treat} \text{-} \text{start} \text{-} \text{clutch}
\quad \text{when} \\
\quad \text{when} \\
\quad \quad \text{motor} \_ \text{actuator} = \text{working} \\
\quad \quad \text{motor} \_ \text{sensor} = \text{working} \\
\quad \quad \text{clutch} \_ \text{actuator} = \text{disengaged} \\
\quad \quad \text{clutch} \_ \text{sensor} = \text{disengaged} \\
\quad \quad \text{door} \_ \text{sensor} = \text{closed} \\
\quad \quad \text{door} \_ \text{actuator} = \text{closed} \\
\quad \quad m = 1 \\
\quad \quad \text{then} \\
\quad \quad \quad \text{clutch} \_ \text{actuator} := \text{engaged} \\
\quad \quad \quad m := 0 \\
\quad \quad \text{end} \\
\quad \text{end}
\end{align*}
a_off
when
  a = 1
  r = 1
  s = 0
  b = 0
  m = 0
then
  a := 0
end

treat_open_door
when
  door_actuator = closed
  door_sensor = closed
  clutch_sensor = disengaged
  clutch_actuator = disengaged
  m = 0
then
  door_actuator := open
end
- **treat_close_door** is the result of depressing button B3
- **treat_stop_clutch** is the result of depressing button B4
- **treat_start_clutch** and **treat_open_door** are automatic
- Environment (no new events)
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - door_close
  - door_open
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button
- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door
Motor
Start
Stop

Clutch
Start
Stop

CONTROLLER

start_motor_impulse
stop_motor_impulse
start_clutch_impulse
stop_clutch_impulse

CLUTCH

motor_actuator

motor_sensor

MOTOR

door_sensordoor_actuator

DOOR

door_actuator
door_sensor

clutch_actuator

clutch_sensor

202
- There are no door buttons

- The door must be closed before engaging the clutch

- The door must be opened after disengaging the clutch

- It is sufficient to connect:
  - button B3 to the door (closing the door)
  - button B4 to the clutch (disengaging the clutch)
- motor_start
- motor_stop
- clutch_start
- clutch_stop
- door_close
- door_open
- push_start_motor_button
- release_start_motor_button
- push_stop_motor_button
- release_stop_motor_button
- push_start_clutch_button
- release_start_clutch_button
- push_stop_clutch_button
- release_stop_clutch_button
- treat_push_start_motor_button
- treat_push_stop_motor_button
- treat_push_stop_motor_button_false
- treat_release_start_motor_button
- treat_release_stop_motor_button
- treat_start_clutch
- treat_stop_clutch
- treat_close_door
- treat_open_door
- treat_close_door_false
- treat_stop_clutch_false
- treat_release_start_clutch_button
- treat_release_stop_clutch_button
- The environment events

- The environment variables modified by environment events

- The sensor variables modified by environment events

- The actuator variables read by environment events

- The controller variables not seen by environment events

- No environment variables in this model
- The controller events

- The controller variables modified by controller events

- The sensor variables read by controller events

- The actuator variables modified by controller events

- The environment variables not seen by controller events

- No environment variables in this model
- 7 sensor variables:
  - motor_sensor
  - clutch_sensor
  - door_sensor
  - start_motor_button
  - stop_motor_button
  - start_clutch_button
  - stop_clutch_button
- **3** actuator variables:
  - `motor_actuator`
  - `clutch_actuator`
  - `door_actuator`

- **5** controller variables (without the counter variables):
  - `start_motor_impulse`
  - `stop_motor_impulse`
  - `start_clutch_impulse`
  - `stop_clutch_impulse`
  - `m`
- 14 environment events,

- 14 controller events,

- 130 lines for environment events,

- 180 lines for controller events.
- 4 weak reactions: 4 buttons (B1, B2, B3, B4)

- 3 strong reactions: 3 devices (motor, clutch, door)

- 3 strong-weak reactions: motor-clutch, clutch-door, motor-door

- 1 strong-strong reaction: clutch-door
Summary: Number of Invariants

- Weak reaction: 6
- Strong reaction: 3
- Strong-weak reaction: 16
- Strong-strong reaction: 7
- Total: 32

- Press (typing): 15
- Press (redundant with those of patterns): 12
- Total: 27
Summary: Number of Proof Obligations

- Weak reaction: 18
- Strong reaction: 12
- Strong-weak reaction: 60
- Strong-strong reaction: 40
- Total: 130

- Press (redundant with those of design patterns): 60

- PO saving: $4 \times 18 + 3 \times 12 + 3 \times 60 + 40 = 328$
- Design patterns: 2 easy interactive, out of 130

- Press: all automatic, out of 60
- 600 lines of C code for the simulation,

- 470 lines come from a direct translation of the last refinement,

- 130 lines correspond to the hand-written interface.
- This design pattern approach *seems to be fruitful*

- It results in a *very systematic* formal development

- Many *other patterns* have to be developed

- *More automation* has to be provided (*plug-in*)