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**
A Typical User Session

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
- Controlling the way the clutch is engaged or disengaged

- Protection by means of a Front Door
The Front Door

open  closed

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
Initial Situation
Starting the Motor: Pressing Button B1
Engaging the Clutch: Pressing Button B3
Disengaging the Clutch: Pressing Button B4
Removing the Part
Putting a Part
Engaging the Clutch: Pressing Button B3
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
The Press works
Disengaging the Clutch: Pressing Button B4
Removing the Part
Removing the Tool
Stopping Motor: Pressing Button B2
Final Situation
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

.
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

- **a** denotes the action
- **r** denotes the reaction
- **ca** and **cr** denote how many times **a** and **r** are set to 1
- **pat0.5** formalizes the weak reaction
Model for weak action and reaction: the Events (1)

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

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

```
r_on
    when
        r = 0
        a = 1
    then
        r := 1
        cr := cr + 1
    end

r_off
    when
        r = 1
        a = 0
    then
        r := 0
    end
```
Summary of Events

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

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

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

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

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

\[
\begin{align*}
\text{pat0.1: } & \quad a \in \{0, 1\} \\
\text{pat0.2: } & \quad r \in \{0, 1\} \\
\text{pat0.3: } & \quad ca \in \mathbb{N} \\
\text{pat0.4: } & \quad cr \in \mathbb{N} \\
\text{pat0.5: } & \quad cr \leq ca
\end{align*}
\]

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

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

\[
\begin{align*}
a\textunderscore off & : \\
\quad \text{when } a & = 1 \\
\quad \text{then} & \\
\quad a & := 0 \\
\quad ca & := ca + 1 \\
\quad \text{end}
\end{align*}
\]

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

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

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

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

- 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**

This is called a **Proof Obligation Rule**
Naming the Proof Obligation

- We have 5 invariants: \texttt{pat0\_1} to \texttt{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

$\vdash$ Modified Invariant

POs are generated by a tool: the POG

$\vdash$ 

a_on

when

$\vdash$ 

then

$\vdash$ 

end

Preservation of invariant $\text{pat0.1}$ by event $\text{a_on}$
The preservation proof of \texttt{pat0.5} ($cr \leq ca$) by event \texttt{r_on} fails

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

\begin{verbatim}
\ldots
  \texttt{cr} \leq \texttt{ca}
  \texttt{r = 0}
  \texttt{a = 1}
\vdash
  \texttt{cr + 1} \leq \texttt{ca}
\end{verbatim}

\begin{verbatim}
\ldots
  \texttt{pat0.5}
  \texttt{guard of r_on}
  \texttt{guard of r_on}
\end{verbatim}

\begin{verbatim}
\ldots
  \texttt{modified pat0.5}
\end{verbatim}
- 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
  cr \leq ca
  r = 0
  a = 1
\vdash
  cr + 1 \leq ca
\end{verbatim}

- We have to add the assumption \(cr < ca\) in our sequent
- The preservation proof of \texttt{pat0.5} \((cr \leq ca)\) by event \texttt{r_on} fails

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

- Two solutions: strengthening the guard or adding a new invariant
First Solution: Strengthening the Guards

```plaintext
r_on
  when
    r = 0
    a = 1
    cr < ca
  then
    r := 1
    cr := cr + 1
end

\ldots
  \cdots
  cr \leq ca
  r = 0
  a = 1
  cr < ca
\vdash
  cr + 1 \leq ca
\ldots
pat0_5
  \vdash
  \text{guard of } r\_on
  \text{guard of } r\_on
\text{new guard of } r\_on
\text{modified } pat0_5

- Drawback: One has to keep variables \textit{cr} and \textit{ca} in the guard
- These variables were introduced for the modelling only
```
- We cannot introduce invariant $cr < ca$ directly (it does not hold)

- We introduce an implicative invariant

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

r_on

when

\[
\begin{align*}
  r &= 0 \\
  a &= 1
\end{align*}
\]

then

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

end

\[
\begin{align*}
  r &= 0 \\
  a &= 1
\end{align*}
\]

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

\[
\begin{align*}
  r &= 0 \\
  a &= 1
\end{align*}
\]

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

- Drawback: One has to prove that this new invariant is maintained
pat0\_6: \quad r = 0 \land a = 1 \Rightarrow cr < ca

cr is incremented
cr<ca
ca is incremented

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

pat0_6: \[ r = 0 \land a = 1 \Rightarrow cr < ca \]

- 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

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

The preservation of this invariant by \( r_{\text{on}} \) leads to proving:

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

\[ r = 0 \land a = 1 \Rightarrow cr < ca \]
\[ r = 0 \]
\[ a = 1 \]
\[ \vdash 1 = 0 \land a = 1 \Rightarrow cr + 1 < ca \]

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 \implies cr < ca
\end{align*}
The counters have been removed.

**init**

```
\text{a} := 0
\text{r} := 0
```

**a\_on**

```
\text{when} \quad \text{a} = 0
\text{then} \quad \text{a} := 1
\text{end}
```

**a\_off**

```
\text{when} \quad \text{a} = 1
\text{then} \quad \text{a} := 0
\text{end}
```

**r\_on**

```
\text{when} \quad \text{r} = 0
\text{then} \quad \text{a} := 1
\text{end}
```

**r\_off**

```
\text{when} \quad \text{r} = 1
\text{then} \quad \text{r} := 0
\text{end}
```
Weak Synchronization of Events
- 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

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

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

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

- Event `a_on` cannot maintain `pat1_1`

---

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

Guard of `a_on`:

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

Modified `pat1_1`
- Event a_on cannot maintain \texttt{pat1\_1}

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

\texttt{ca} \leq \texttt{cr}
\texttt{cr} \leq \texttt{ca}
\texttt{ca} \leq \texttt{cr} + 1
\texttt{a} = 0
\texttt{a} = 0
\texttt{ca} + 1 \leq \texttt{cr} + 1
\end{verbatim}

- We need the assumption \texttt{ca} \leq \texttt{cr}
- Event `a_on` cannot maintain `pat1_1`

```
a_on
  when
   a = 0
  then
   a := 1
   ca := ca + 1
end
```

- We need the assumption `ca ≤ cr`
- But we already have assumption `cr ≤ ca` (this is `pat0_5`)

\[
cr \leq ca
\]
\[
ca \leq cr + 1
\]
\[
a = 0
\]
\[
\vdash
\]
\[
ca + 1 \leq cr + 1
\]

Modified `pat1_1`
- Event a_on cannot maintain **pat1.1**

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
end
```

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

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

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

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

- 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

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

\[
\begin{aligned}
a = 0 & \implies cr = ca \\
a = 1 & \\
0 = 0 & \implies cr = ca
\end{aligned}
\]

\[\text{pat1.2} \quad \text{guard of a\_off} \quad \text{modified pat1.2} \]

- This suggests a new invariant:

\[ a = 1 \implies cr = ca \]
Proposing a new invariant

We need: \( a = 1 \Rightarrow cr = ca \)

- But we already have the following:

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

This suggests the following:

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

```plaintext
\[
\begin{array}{l}
a_{\text{off}} \\
\text{when} \\
\quad a = 1 \\
\quad \text{then} \\
\quad a := 0 \\
\end{array}
\]

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

\[\vdash 0 = 0 \Rightarrow cr = ca\] modified \( \text{pat1}_2 \)

- 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{center}
\begin{tabular}{|c|c|c|}
\hline
& $a = 0 \Rightarrow cr = ca$ & \texttt{pat1\_2} \\
\hline
& $a = 1 \land r = 1 \Rightarrow cr = ca$ & \texttt{pat1\_3} \\
\hline
& $r = 1$ & new guard \\
\hline
& $a = 1$ & guard \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
& $a = 1$ & \texttt{pat1\_2} \\
\hline
& $r = 1$ & \texttt{modified pat1\_2} \\
\hline
\end{tabular}
\end{center}

- We need to strengthen its guard because of \texttt{pat1\_3}:
Problems with the New Invariant Preservation

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

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

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

- Problem with \text{a\_on} since \( a \) becomes 1 and \( ca \) is incremented
- No problem with \text{a\_off} since \( a \) becomes 0
- Problem with \text{r\_on} since \( a = 1 \) (guard), \( r \) becomes 1, and \( cr \) is incremented
- No problem with \text{r\_off} since \( r \) becomes 0
- Event \( a_{\text{on}} \) cannot maintain invariant \( \text{pat1.3} \)

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

\begin{verbatim}
a_{\text{on}}
  when
    \( a = 0 \)
  then
    \( a := 1 \)
    \( ca := ca + 1 \)
  end
\end{verbatim}

- This suggest strengthening the guard of \( a_{\text{on}} \): \( r = 0 \)
- Event \( a_{\text{on}} \) cannot maintain invariant \( \text{pat1.3} \)

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

```
a_{\text{on}}
  \text{when} \\
  a = 0 \\
  r = 0 \\
  \text{then} \\
  \text{a := 1} \\
  ca := ca + 1 \\
end
```

- This suggests strengthening the guard of \( a_{\text{on}} \): \( r = 0 \)
Problems with the New Invariant Preservation

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

- 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 pat1_3

- Invariant pat1_3

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

```
<table>
<thead>
<tr>
<th>r_on</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
<td>r = 0</td>
<td>r = 0</td>
</tr>
<tr>
<td></td>
<td>a = 1</td>
<td>a = 1</td>
</tr>
<tr>
<td>then</td>
<td>r := 1</td>
<td>r := 1</td>
</tr>
<tr>
<td></td>
<td>cr := cr + 1</td>
<td>cr := cr + 1</td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

\[ a = 1 \land r = 1 \Rightarrow cr = ca \]
\[ r = 0 \]
\[ a = 1 \]

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

We have forgotten invariants pat1_1 and pat0_6
What about Event r\_on and Invariant pat1\_3

- Invariant \textbf{pat1\_3}

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

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

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

\[ ca \leq cr + 1 \]

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

\[ r = 0 \]

\[ a = 1 \]

\[ \vdash a = 1 \land 1 = 1 \implies cr + 1 = ca \]

- 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.

**init**

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

**a_on**

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

**a_off**

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

**r_on**

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

**r_off**

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

\[ a_{on} \leftrightarrow a_{off} \]
\[ r_{on} \leftrightarrow r_{off} \]
Merging the invariants

- Putting together these two invariants

\[
\text{pat1.2: } a = 0 \implies ca = cr
\]

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

- leads to the following

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

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

This can be simplified to

\begin{align*}
\text{pat2.1:} & \quad a = 1 \land r = 0 \implies ca = cr + 1 \\
\text{pat2.2:} & \quad a = 0 \lor r = 1 \implies ca = cr
\end{align*}
Summary of the State for the **Strong 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{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*}
Comparing the two Invariants

\[\text{pat2}_1: \quad a = 1 \land r = 0 \implies ca = cr + 1\]

\[\text{pat2}_2: \quad a = 0 \lor r = 1 \implies ca = cr\]

cr is incremented

cia is incremented

<table>
<thead>
<tr>
<th>pat1_4</th>
<th>pat0_6</th>
<th>pat1_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=0</td>
<td>a=1</td>
<td>r=1</td>
</tr>
<tr>
<td>ca = cr</td>
<td>r=0</td>
<td>ca = cr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ca=cr+1</td>
</tr>
</tbody>
</table>
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
## Requirements: Describing Equipment

<table>
<thead>
<tr>
<th>The system has got the following pieces of equipment: a Motor, a Clutch, and a Door</th>
<th>EQP_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four Buttons are used to start and stop the motor, and engage and disengage the clutch</td>
<td>EQP_2</td>
</tr>
<tr>
<td>A Controller is supposed to manage this equipment</td>
<td>EQP_3</td>
</tr>
<tr>
<td>Requirements: Connection Constraints</td>
<td></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>
</tbody>
</table>
### Requirements: Relationship Between Motor and Clutch

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<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>Code</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>
4. Proposing a Refinement Strategy
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
Controller and Motor are strongly synchronized.

Controller | Motor
---|---
Controller are Equipment are strongly synchronized | FUN_2
Model for strong action and reaction: the Final Events

The counters have been removed

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

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

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

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

\begin{align*}
r_{off} \quad & \text{when} \\
& \quad r = 1 \\
& \quad a = 0 \\
& \quad \text{then} \\
& \quad r := 0 \\
& \quad \text{end}
\end{align*}
set: \( \text{STATUS} \)
constants: \( \text{stopped, working} \)

\[
\begin{align*}
\text{axm0.1:} & \quad \text{STATUS} = \{\text{stopped, working}\} \\
\text{axm0.2:} & \quad \text{stopped} \neq \text{working}
\end{align*}
\]
Initial Model: the State

variables:  

\begin{itemize}
  \item motor_actuator
  \item motor_sensor
\end{itemize}

\begin{itemize}
  \item inv0_1:  motor_sensor \in STATUS
  \item inv0_2: motor_actuator \in STATUS
\end{itemize}
Initial Model: the Synchronization

Controller → Action → Motor

motor_actuator

Motor → Strong Reaction → motor_sensor

Strong Reaction
- We instantiate the weak pattern as follows:

\[
\begin{align*}
  a & \mapsto \text{motor\_actuator} \\
  r & \mapsto \text{motor\_sensor} \\
  0 & \mapsto \text{stopped} \\
  1 & \mapsto \text{working}
\end{align*}
\]

\[
\begin{align*}
  \text{a\_on} & \mapsto \text{treat\_start\_motor} \\
  \text{a\_off} & \mapsto \text{treat\_stop\_motor} \\
  \text{r\_on} & \mapsto \text{Motor\_start} \\
  \text{r\_off} & \mapsto \text{Motor\_stop}
\end{align*}
\]

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

\[
\text{init} \quad \\
\begin{align*}
a & := 0 \\
r & := 0
\end{align*}
\]
Initial Model: Controller Events (1)

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

\[
treat_{\text{start}\_\text{motor}}
\begin{align*}
\text{when} & \quad \text{motor}_{\text{actuator}} = \text{stopped} \\
& \quad \text{motor}_{\text{sensor}} = \text{stopped} \\
\text{then} & \quad \text{motor}_{\text{actuator}} := \text{working} \\
\end{align*}
\]
Initial Model: Environment Event (1)

Initial Model:

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

Motor start:

- **when**
  - motor sensor = stopped
  - motor actuator = working
  - **then**
    - motor sensor := working
  - **end**
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
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
- Environment
  - motor_start
  - motor_stop

- Controller
  - treat_start_motor
  - treat_stop_motor
1st Ref.: Connecting the Motor Buttons to the Controller

Buttons and Controller are weakly synchronized    FUN_1
The counters have been removed.

Model for weak action and reaction: the Final Events

init
   \( a := 0 \)
   \( r := 0 \)

\( \text{a\_on} \)
 when
   \( a = 0 \)
 then
   \( a := 1 \)
 end

\( \text{a\_off} \)
 when
   \( a = 1 \)
 then
   \( a := 0 \)
 end

\( \text{r\_on} \)
 when
   \( r = 0 \)
 then
   \( a := 1 \)
   \( r := 1 \)
 end

\( \text{r\_off} \)
 when
   \( r = 1 \)
 then
   \( a := 0 \)
   \( r := 0 \)
 end
variables:  
\[
\begin{align*}
\text{inv1}_1: & \quad \text{stop}_\text{motor}_\text{button} \in \text{BOOL} \\
\text{inv1}_2: & \quad \text{start}_\text{motor}_\text{button} \in \text{BOOL} \\
\text{inv1}_3: & \quad \text{stop}_\text{motor}_\text{impulse} \in \text{BOOL} \\
\text{inv1}_4: & \quad \text{start}_\text{motor}_\text{impulse} \in \text{BOOL}
\end{align*}
\]
First Refinement: the State

```
<table>
<thead>
<tr>
<th>Start Button</th>
<th>Stop Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>action</td>
</tr>
<tr>
<td>start_motor_button</td>
<td>stop_motor_button</td>
</tr>
<tr>
<td>start_motor_impulse</td>
<td>stop_motor_impulse</td>
</tr>
<tr>
<td>weak reaction</td>
<td>weak reaction</td>
</tr>
<tr>
<td>CONTROLLER</td>
<td></td>
</tr>
</tbody>
</table>
```
- We instantiate the pattern as follows:

\[
a \mapsto \textit{start\_motor\_button}
\]

\[
r \mapsto \textit{start\_motor\_impulse}
\]

\[
0 \mapsto \text{FALSE}
\]

\[
1 \mapsto \text{TRUE}
\]

\[
a\_on \mapsto \text{push\_start\_motor\_button}
\]

\[
a\_off \mapsto \text{release\_stop\_motor\_button}
\]

\[
r\_on \mapsto \text{treat\_push\_start\_motor\_button}
\]

\[
r\_off \mapsto \text{treat\_release\_start\_motor\_button}
\]

- We rename \textit{treat\_start\_motor} as \textit{treat\_push\_start\_motor\_button}
init

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

\begin{align*}
\text{motor}_\text{actuator} &:= \text{stopped} \\
\text{motor}_\text{sensor} &:= \text{stopped} \\
\text{start}_\text{motor}_\text{button} &:= \text{FALSE} \\
\text{start}_\text{motor}_\text{impulse} &:= \text{FALSE}
\end{align*}
```
First Refinement: New Environment Events (1)

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
```
r_on

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

\( \text{treat\_push\_start\_motor\_button} \)
\( \text{refines} \)
\( \text{treat\_start\_motor} \)

when
  \( \text{start\_motor\_impulse} = \text{FALSE} \)
  \( \text{start\_motor\_button} = \text{TRUE} \)
  \( \text{motor\_actuator} = \text{stopped} \)
  \( \text{motor\_sensor} = \text{stopped} \)
then
  \( \text{start\_motor\_impulse} := \text{TRUE} \)
  \( \text{motor\_actuator} := \text{working} \)
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
- Refinement 1
- Refinement n
- Instantiated Pattern

Refines relationships shown in the diagram.
r_off

<table>
<thead>
<tr>
<th>when</th>
<th>r = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a = 0</td>
</tr>
<tr>
<td>then</td>
<td>r := 0</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>treat_release_start_motor_button</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>start_motor_impulse := TRUE</td>
</tr>
<tr>
<td>start_motor_button := FALSE</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>start_motor_impulse := FALSE</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
- We instantiate the pattern as follows:

\[
\begin{align*}
  a & \rightsquigarrow \text{stop\_motor\_button} \\
  r & \rightsquigarrow \text{stop\_motor\_impulse} \\
  0 & \rightsquigarrow \text{FALSE} \\
  1 & \rightsquigarrow \text{TRUE} \\
  \text{a\_on} & \rightsquigarrow \text{push\_stop\_motor\_button} \\
  \text{a\_off} & \rightsquigarrow \text{release\_stop\_motor\_button} \\
  \text{r\_on} & \rightsquigarrow \text{treat\_push\_stop\_motor\_button} \\
  \text{r\_off} & \rightsquigarrow \text{treat\_release\_stop\_motor\_button}
\end{align*}
\]
init

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

init

\[
\begin{align*}
motor\_actuator & := \text{stopped} \\
motor\_sensor & := \text{stopped} \\
start\_motor\_button & := \text{FALSE} \\
start\_motor\_impulse & := \text{FALSE} \\
stop\_motor\_button & := \text{FALSE} \\
stop\_motor\_impulse & := \text{FALSE}
\end{align*}
\]
a_on
   when
   a = 0
   then
      a := 1
   end

push_stop_motor_button
   when
      stop_motor_button = FALSE
   then
      stop_motor_button := TRUE
   end

a_off
   when
   a = 1
   then
      a := 0
   end

release_stop_motor_button
   when
      stop_motor_button = TRUE
   then
      stop_motor_button := FALSE
   end
r_on

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

treat_push_stop_motor_button

refines
treat_stop_motor

when
  stop_motor_impulse = FALSE
  stop_motor_button = TRUE

then
  motor_sensor = working
  motor_actuator = working
  stop_motor_impulse := TRUE
  motor_actuator := stopped
end
First Refinement: New Controller Events (2)

\[
\text{r\_off} \\
\text{\hspace{1cm} when} \\
\text{\hspace{2cm} r = 1} \\
\text{\hspace{2cm} a = 0} \\
\text{\hspace{1cm} then} \\
\text{\hspace{2cm} r := 0} \\
\text{\hspace{1cm} end}
\]

\[
\text{treat\_release\_stop\_motor\_button} \\
\text{\hspace{1cm} when} \\
\text{\hspace{2cm} stop\_motor\_impulse = TRUE} \\
\text{\hspace{2cm} stop\_motor\_button = FALSE} \\
\text{\hspace{1cm} then} \\
\text{\hspace{2cm} stop\_motor\_impulse := FALSE} \\
\text{\hspace{1cm} end}
\]
Independent Synchronizations

push_start_motor_button \rightarrow release_start_motor_button

push_start_motor_button \rightarrow release_start_motor_button

treat_push_start_motor_button \rightarrow treat_release_start_motor_button

treat_push_start_motor_button \rightarrow 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_release_stop_motor_button
- treat_push_stop_motor_button

- release_stop_motor_button
- push_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`
Weak and Strong Reactions Together

Start Button  Stop Button

action

start_motor_button  stop_motor_button

start_motor_impulse  stop_motor_impulse

weak reaction  weak reaction

CONTROLLER

motor_actuator

motor_sensor

MOTOR

action

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

  \[ \neg (motor\_actuator = stopped \land motor\_sensor = stopped) \]

- We need another event
Problems with treat_push_start_motor_button

- 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 treat_push_stop_motor_button

- 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:

\[ \text{stop\_motor\_impulse} := \text{TRUE} \]
First Refinement: Summary of the Events (1)

- 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

start_motor_impulse

stop_motor_impulse

CONTROLLER

motor_actuator

motor_sensor

CLUTCH

MOTOR
- We introduce the set in a new context:

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

- We copy the initial model where we instantiate:

\[
\begin{align*}
\text{motor} & \mapsto \text{clutch} \\
\text{STATUS} & \mapsto \text{CLUTCH} \\
\text{working} & \mapsto \text{engaged} \\
\text{stopped} & \mapsto \text{disengaged}
\end{align*}
\]
- 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 | SAF₁ |

- 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
The Synchronization is Weak (1)

When the clutch is disengaged,
then
the motor can be started and stopped several times.
When the motor works,

then

the clutch can be engaged and disengaged several times.
Putting the Two Together
Synchronizing the Reactions Without Touching them

![Diagram showing the synchronization of reactions without touching them]

- a_{on} \leftrightarrow a_{off}
- r_{on} \leftrightarrow r_{off}
- b_{on} \leftrightarrow b_{off}
- s_{on} \leftrightarrow s_{off}
The Initial State Situation

\[
\begin{align*}
\text{dbl0\_1:} & \quad a \in \{0, 1\} \\
\text{dbl0\_2:} & \quad r \in \{0, 1\} \\
\text{dbl0\_3:} & \quad ca \in \mathbb{N} \\
\text{dbl0\_4:} & \quad cr \in \mathbb{N} \\
\text{dbl0\_5:} & \quad a = 1 \land r = 0 \Rightarrow ca = cr + 1 \\
\text{dbl0\_6:} & \quad a = 0 \lor r = 1 \Rightarrow ca = cr \\
\text{dbl0\_7:} & \quad b \in \{0, 1\} \\
\text{dbl0\_8:} & \quad s \in \{0, 1\} \\
\text{dbl0\_9:} & \quad cb \in \mathbb{N} \\
\text{dbl0\_10:} & \quad cs \in \mathbb{N} \\
\text{dbl0\_11:} & \quad b = 1 \land s = 0 \Rightarrow cb = cs + 1 \\
\text{dbl0\_12:} & \quad b = 0 \lor s = 1 \Rightarrow cb = cs
\end{align*}
\]
a_on
when
    a = 0
    r = 0
then
    a := 1
    ca := ca + 1
end

r_on
when
    r = 0
    a = 1
then
    r := 1
    cr := cr + 1
end

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

r_off
when
    r = 1
    a = 0
then
    r := 0
end
The Initial Event Situation (2)

\begin{align*}
\text{b\_on} & \quad \text{b\_off} \\
\text{when} & \quad \text{when} \\
\quad b = 0 & \quad b = 1 \\
\quad s = 0 & \quad s = 1 \\
\text{then} & \quad \text{then} \\
\quad b := 1 & \quad b := 0 \\
\quad cb := cb + 1 & \quad cs := cs + 1 \\
\end{align*}

\begin{align*}
\text{s\_on} & \quad \text{s\_off} \\
\text{when} & \quad \text{when} \\
\quad s = 0 & \quad s = 1 \\
\quad b = 1 & \quad b = 0 \\
\text{then} & \quad \text{then} \\
\quad s := 1 & \quad s := 0 \\
\end{align*}
The Synchronizing Invariant

\(\text{dbl1}_1: \quad s = 1 \implies r = 1\)

- It seems sufficient to add the following guards

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

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

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

- We introduce the following additional invariants

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

\[
\text{dbl1}_3: \quad a = 0 \implies s = 0
\]
Maintaining Invariant dbl1\_2 (1)

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

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

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

\[
\begin{align*}
\text{b\_on} \\
\text{when} \\
\quad b = 0 \\
\quad s = 0 \\
\quad r = 1 \\
\text{then} \\
\quad b := 1 \\
\quad cb := cb + 1 \\
\text{end}
\end{align*}
\]
Maintaining (Contraposition of) Invariant \( \text{dbl1}_2 \) (2)

\[
\text{dbl1}_2: \quad b = 1 \implies r = 1 \quad (r = 0 \implies b = 0)
\]

In order to maintain this invariant, we have to refine \( r_{\text{off}} \)

\[
\begin{array}{l}
\text{r}_{\text{off}} \\
\quad \text{when} \\
\qquad r = 1 \\
\quad a = 0 \\
\quad \text{then} \\
\qquad r := 0 \\
\quad \text{end}
\end{array}
\]

\[
\begin{array}{l}
\text{r}_{\text{off}} \\
\quad \text{when} \\
\qquad r = 1 \\
\quad a = 0 \\
\quad b = 0 \\
\quad \text{then} \\
\qquad r := 0 \\
\quad \text{end}
\end{array}
\]

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

- We introduce the following invariant

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

\[
\text{dbl1\_4: } \quad a = 0 \implies b = 0
\]
Maintaining Invariant \texttt{dbl1\_3} (1)

\[ \texttt{dbl1\_3: } a = 0 \implies s = 0 \]

In order to maintain this invariant, we have to refine \texttt{a\_off}

\begin{verbatim}
\texttt{a\_off}
\texttt{when}
\quad a = 1
\quad r = 1
\texttt{then}
\quad a := 0
\texttt{end}
\end{verbatim}

\begin{verbatim}
\texttt{a\_off}
\texttt{when}
\quad a = 1
\quad r = 1
\quad s = 0
\texttt{then}
\quad a := 0
\texttt{end}
\end{verbatim}
Maintaining (Contraposition of) Invariant dbl1_3 (2)

dbl1_3: \( a = 0 \Rightarrow s = 0 \quad (s = 1 \Rightarrow a = 1) \)

In order to maintain this invariant, we have to refine \( s \_on \)

```
when
  s = 0
  b = 1
then
  s := 1
  cs := cs + 1
end
```

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

```plaintext
s_on
  when
    s = 0
    b = 1
    a = 1
  then
    s := 1
    cs := cs + 1
  end

- We have to introduce the following invariant

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

- Fortunately, this is \textit{dbl1}\_4 \((a = 0 \Rightarrow b = 0)\) contraposed
Maintaining Invariant dbl1_4 (1)

\[
\text{dbl1}_4: \quad a = 0 \Rightarrow b = 0
\]

In order to maintain this invariant, we have to **refine** a\_off again

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

\[
\begin{align*}
\text{a\_off} & \\
\text{when} & \\
\quad a = 1 & \\
\quad r = 1 & \\
\quad s = 0 & \\
\quad b = 0 & \\
\text{then} & \\
\quad a := 0 & \\
\text{end} &
\end{align*}
\]
Maintaining (Contraposition of) Invariant `dbl1_4` (2)

\[
\text{dbl1}_4: \quad a = 0 \Rightarrow b = 0 \quad (b = 1 \Rightarrow a = 1)
\]

In order to maintain this invariant, we have to refine `b_on` again.

```
b_on
when
  b = 0
  s = 0
  r = 1
then
  b, cb := 1, cb + 1
end
```

```
b_on
when
  b = 0
  s = 0
  r = 1
then
  a = 1
  b, cb := 1, cb + 1
end
```
Summary of Refinement: Reactions have not been Touched

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

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

\[
\begin{align*}
\text{a\_off} & \\
\text{when} & \\
& \quad a = 1 \\
& \quad r = 1 \\
& \quad s = 0 \\
& \quad b = 0 \\
\text{then} & \\
& \quad a := 0 \\
\text{end} & \\
\end{align*}
\]
Intuition about the Invariants

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

This can be put into a single invariant

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

with the following contraposited form

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

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

dbl1_5: \[ b = 1 \lor s = 1 \implies a = 1 \land r = 1 \]

dbl1_6: \[ a = 0 \lor r = 0 \implies b = 0 \land s = 0 \]
Weak Synchronization of Strong Reaction: the Problem

Diagram:

- $a_{on}$ to $a_{off}$
- $r_{on}$ to $r_{off}$
- $b_{on}$ to $b_{off}$
- $s_{on}$ to $s_{off}$
Weak Synchronization of Strong Reaction: the Solution

![Diagram showing the synchronization of reactions with labels a_on, a_off, r_on, r_off, b_on, b_off, s_on, s_off.](attachment:image.png)
When the clutch is engaged, the motor must work

<table>
<thead>
<tr>
<th>SAF_1</th>
</tr>
</thead>
</table>

**inv3_1:**

\[
\text{clutch}_{\text{sensor}} = \text{engaged} \\
\Rightarrow \\
\text{motor}_{\text{sensor}} = \text{working}
\]

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

\[ a \rightsquigarrow \text{motor\_actuator} \quad a\text{\_on} \rightsquigarrow \text{treat\_push\_start\_motor\_button} \]
\[ r \rightsquigarrow \text{motor\_sensor} \quad a\text{\_off} \rightsquigarrow \text{treat\_push\_stop\_motor\_button} \]
\[ 0 \rightsquigarrow \text{stopped} \quad r\text{\_on} \rightsquigarrow \text{Motor\_start} \]
\[ 1 \rightsquigarrow \text{working} \quad r\text{\_off} \rightsquigarrow \text{Motor\_stop} \]

\[ b \rightsquigarrow \text{clutch\_actuator} \quad b\text{\_on} \rightsquigarrow \text{treat\_start\_clutch} \]
\[ s \rightsquigarrow \text{clutch\_sensor} \quad b\text{\_off} \rightsquigarrow \text{treat\_stop\_clutch} \]
\[ 0 \rightsquigarrow \text{disengaged} \quad s\text{\_on} \rightsquigarrow \text{Clutch\_start} \]
\[ 1 \rightsquigarrow \text{engaged} \quad s\text{\_off} \rightsquigarrow \text{Clutch\_stop} \]
Translating the pattern invariants (1)

\[
\begin{align*}
\text{dbl1}_1: & \quad s = 1 \implies r = 1 \\
\text{dbl1}_2: & \quad b = 1 \implies r = 1
\end{align*}
\]

\[
\begin{align*}
\text{inv3}_1: & \quad \text{clutch}_\text{sensor} = \text{engaged} \quad \implies \quad \text{motor}_\text{sensor} = \text{working} \\
\text{inv3}_2: & \quad \text{clutch}_\text{actuator} = \text{engaged} \quad \implies \quad \text{motor}_\text{sensor} = \text{working}
\end{align*}
\]
Translating the pattern invariants (2)

\[
\begin{align*}
\text{dbl1}_3: & \quad a = 0 \quad \Rightarrow \quad s = 0 \\
\text{dbl1}_4: & \quad a = 0 \quad \Rightarrow \quad b = 0
\end{align*}
\]

\[
\begin{align*}
\text{inv3}_3: & \quad \Rightarrow \quad \text{motor}_\text{actuator} = \text{stopped} \\
& \quad \Rightarrow \quad \text{clutch}_\text{sensor} = \text{disengaged}
\end{align*}
\]

\[
\begin{align*}
\text{inv3}_4: & \quad \Rightarrow \quad \text{motor}_\text{actuator} = \text{stopped} \\
& \quad \Rightarrow \quad \text{clutch}_\text{actuator} = \text{disengaged}
\end{align*}
\]
Adapting the Events of the Pattern (1)

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

\[
\begin{align*}
\text{treat_start_clutch} & \\
\text{when} & \\
\quad \text{clutch_actuator} = \text{disengaged} & \\
\quad \text{clutch_sensor} = \text{disengaged} & \\
\quad \text{motor_sensor} = \text{working} & \\
\quad \text{motor_actuator} = \text{working} & \\
\text{then} & \\
\quad \text{clutch_actuator} := \text{engaged} & \\
\text{end} & \\
\end{align*}
\]
<table>
<thead>
<tr>
<th>a_off</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
</tr>
<tr>
<td>a = 1</td>
</tr>
<tr>
<td>r = 1</td>
</tr>
<tr>
<td>s = 0</td>
</tr>
<tr>
<td>b = 0</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td>a := 0</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

```plaintext
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
```

Adapting the events of the pattern (2)
- 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
- We copy (after renaming "motor" to "door") what has been done in the initial model
- We introduce the set in a new context:

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

- We copy the initial model where we instantiate:

\[ \text{motor} \leadsto \text{door} \]
\[ \text{STATUS} \leadsto \text{DOOR} \]
\[ \text{working} \leadsto \text{closed} \]
\[ \text{stopped} \leadsto \text{open} \]
Fourth Refinement: Summary of the Events (1)

- 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)

<table>
<thead>
<tr>
<th>When the motor is stopped, the door must be open</th>
<th>SAF_3</th>
</tr>
</thead>
</table>
- 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 door is closed, the motor must work | SAF_3’ |

- SAF_3’ is the contraposited 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:

<table>
<thead>
<tr>
<th>When the clutch is engaged, the motor must work</th>
<th>SAF_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>SAF_1</td>
<td>When the <strong>clutch</strong> is engaged, the <strong>motor</strong> must work</td>
</tr>
<tr>
<td>SAF_2</td>
<td>When the <strong>clutch</strong> is engaged, the <strong>door</strong> must be closed</td>
</tr>
<tr>
<td>SAF_3’</td>
<td>When the <strong>door</strong> is closed, the <strong>motor</strong> must work</td>
</tr>
</tbody>
</table>

- Requirement SAF_1 is now redundant: \( \text{SAF}_2 \land \text{SAF}_3' \Rightarrow \text{SAF}_1 \)
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

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

- 6th (7th) refinement: Connecting the clutch button to the controller
- Environment \textit{(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.
- When the door is closed, the clutch cannot be disengaged several times.
What we want:

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

\[ cr = cs \quad \lor \quad cr = cs + 1 \]
How about counters $ca$ and $cb$?
In Search of a Solution

\[ ca = cb + 1 \]

\[ a = 1 \]

\[ b = 0 \]

\[ a = 1 \text{ and } b = 0 \]
a = 1 \land b = 0 \Rightarrow ca = cb + 1
This Solution Does not Work

\[ a = 1 \quad \text{and} \quad b = 0 \]

\[ ca = cb + 1 \quad \text{ca} = cb \]

\[ a = 1 \quad \text{and} \quad b = 0 \]
\[ m = 1 \Rightarrow ca = cb + 1 \]
\[ m = 0 \Rightarrow ca = cb \]
The Events

\[ m = 0 \]

\[ m = 1 \]

a\_on

b\_on

a\_off

m = 1

m = 0

m = 0
The Modified Events

a_on
when
  a = 0
  r = 0
then
  a := 1
  ca := ca + 1
  m := 1
end

b_on
when
  r = 1
  a = 1
  b = 0
  s = 0
  m = 1
then
  b := 1
  cb := cb + 1
  m := 1
end

a_off
when
  a = 1
  r = 1
  b = 0
  s = 0
  m = 0
then
  a := 0
end
How about counters *cr* and *cs*
In Search of a Solution

\[ m = 0 \]

\[ m = 1 \]

\[ r = 1 \]

\[ s = 0 \]

\[ cr = cs \]

\[ cr = cs + 1 \]

\[ cr = cs \]

\[ m = 1 \]

\[ s = 0 \]

\[ m = 0 \]

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

\[ m = 0 \]

\[ m = 1 \]

\[ c_r = c_s \]

\[ c_r = c_s + 1 \]

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

\[ m = 0, \quad c_r = c_s, = c_s + 1, \quad c_r = c_s \]

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

\[ m = 1, \quad b = 1 \]

\[ m = 1, \quad s = 0 \]

\[ m = 0 \]

\[ b = 0, \quad m = 0 \]
The Solution

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

\[ r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \implies cr = cs \]
dbl2_1: \( m \in \{0, 1\} \)

dbl2_2: \( m = 1 \Rightarrow ca = cb + 1 \)

dbl2_3: \( m = 0 \Rightarrow ca = cb \)

dbl2_4: \( r = 1 \land s = 0 \land (m = 1 \lor b = 1) \Rightarrow cr = cs + 1 \)

dbl2_5: \( r = 0 \lor s = 1 \lor (m = 0 \land b = 0) \Rightarrow cr = cs \)
- The following theorems are easy to prove

\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*}

\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

- The two new invariants were discovered while doing the proof
- The proofs are now completely automatic
Instantiation

- door closed
- clutch engaged
- treat_open_door (a_off)
- treat_close_door (a_on)
- treat_start_clutch (b_on)
- We instantiate the pattern as follows:

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

a\_on $\sim$ treat\_close\_door

a\_off $\sim$ treat\_open\_door

b\_on $\sim$ treat\_start\_clutch
6th Refinement: Adapting the events of the pattern (2)

\[
\text{a_on} \\
\text{when} \\
\hspace{1em} a = 0 \\
\hspace{1em} r = 0 \\
\text{then} \\
\hspace{1em} a := 1 \\
\hspace{1em} m := 1 \\
\text{end}
\]

\[
\text{treat_close_door} \\
\hspace{1em} \text{when} \\
\hspace{2em} \text{door_actuator} = \text{open} \\
\hspace{2em} \text{door_sensor} = \text{open} \\
\hspace{2em} \text{motor_actuator} = \text{working} \\
\hspace{2em} \text{motor_sensor} = \text{working} \\
\hspace{1em} \text{then} \\
\hspace{2em} \text{door_actuator} := \text{closed} \\
\hspace{2em} m := 1 \\
\text{end}
\]
b_on
when

\begin{align*}
b &= 0 \\
s &= 0 \\
r &= 1 \\
a &= 1 \\
m &= 1
\end{align*}
then
\begin{align*}
b &:= 1 \\
m &:= 0
\end{align*}
end

treat_start_clutch

when

\begin{align*}
\text{motor\_actuator} &= \text{working} \\
\text{motor\_sensor} &= \text{working} \\
\text{clutch\_actuator} &= \text{disengaged} \\
\text{clutch\_sensor} &= \text{disengaged} \\
\text{door\_sensor} &= \text{closed} \\
\text{door\_actuator} &= \text{closed} \\
m &= 1
\end{align*}
then
\begin{align*}
\text{clutch\_actuator} &:= \text{engaged} \\
m &:= 0
\end{align*}
end
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
7th Reft.: Connecting the Controller to the Clutch Buttons

Motor
- start_motor_impulse
- stop_motor_impulse
- start_clutch_impulse
- stop_clutch_impulse

Clutch
- start
- stop

CLUTCH
- clutch_actuator
- clutch_sensor

MOTOR
- motor_actuator
- motor_sensor

DOOR
- door_actuator
- door_sensor

CONTROLLER
- 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)
Seventh Refinement: Summary of the Events (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
- push_start_clutch_button
- release_start_clutch_button
- push_stop_clutch_button
- release_stop_clutch_button
- 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
- treat_close_door_false
- treat_stop_clutch_false
- treat_release_start_clutch_button
- treat_release_stop_clutch_button
Decomposing the Final Model: Environment

- 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
Decomposing the Final Model: Controller

- 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
Summary: Events of the Last Refinement

- 14 environment events,

- 14 controller events,

- 130 lines for environment events,

- 180 lines for controller events.
Summary: Usage of the Design Patterns

- 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
- 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)