Università
degli Studi
di Padova

# University of Padua

Department of Mathematics “Tullio Levi-Civita”

Formal Methods for Cyber-Physical Systems

# ASSIGNMENT 2

Luca Monetti (Mat. 2199440)

# Introduction and objectives

The primary objective of the second assignement is the design and implementation of a supervisory control system for a virtual production plant using the Eclipse ESCET toolset with the CIF modeling language. The plant is composed by 3 machines, 3 depots and two rover that moves inside the plant. The role of each component will be described in the next sections.

## Structure of the report

The present report documents the design and implementation process of the supervisory control system.

The report is structured as follows:

- In Section 2 the architecture of the plant is described, detailing the role of each component.
- In Section 3 the graphic interface of the plant is illustrated
- In Section 4 the requirements of the plant are listed and explained.
- In Section 5 the tooldef file is explained.
- Finally, in Section 6 some conclusions are drawn.

# Plant Architecture

Each component of the plant has a specific role in the production process:

- **Machines**: There are three machines (M1, M2, M3) responsible for processing raw materials into finished products.
- **Depots**: The plant includes three depots (D1, D2, D3) for storing the workitems.
- **Rovers**: Two rovers (BR, OR) can freely move within the plant using a battery consumed on each movement. The BR can carry 2 workitem of each type and move them within the machines. The OR can repair the machines when they break down.

## Machine modeling

Each machine is modeled as a state machine with three states that represent its operational status:

- **Idle**: The machine is ready to process workitems. This is the initial and marked state.
- **Working**: The machine is processing the workitem after the `MX_start` events. From this state, the machine can either complete the processing and return to the **Idle** state upon the `M1_finish_success` event or transition to the **Broken** state if the `MX_broken` event occurs.
- **Broken**: The machine is out of order and needs to be repaired. The machine remains locked until the `MX_repaired` event occurs, which is synchronized with the Orange Rover's presence at the machine coordinates.

## Depot modeling

Each depot is modeled as an extended state machine with a discrete variable `disc int[0..2] quantity` that tracks the number of workitems stored in the depot.

All the transition start from the same initial and marked state. Each depot increments the `quantity` variable upon the corresponding `MX_finish_success` events and decremnents it upon the `BR_take_X` events, ensuring that the depot's capacity constraints are respected.

## Rovers modeling

The rovers are modeled as extended state machines with 3 discrete variables:

- `disc int[0..8] energy`: Represents the rover's battery level, which decreases with each movement and can be recharged at the Charging Stations.

- `disc int[1..6] x`: Represents the rover's current x-coordinate within the plant.
- `disc int[1..4] y`: Represents the rover's current y-coordinate within the plant.

Each rover can move in four directions (up, down, left, right) using the corresponding movement events:
- `XR_move_up`
- `XR_move_down`
- `XR_move_left`
- `XR_move_right`

The rovers can only move if they have sufficient energy and are within the plant's boundaries.

To charge the rover's battery, the `XR_charge` event is used when the rover is at one of the two Charging Stations available. This event resets the `energy` variable to its maximum value of 8.

**Blue Rover (BR)**
The Blue Rover (BR) is responsible for transporting workitems between depots and machines. It can carry up to 2 workitems of each type.

This rover can pick up workitems from depots and the source using the `BR_take_X` events, the rover can only pick up a workitem from the same cell if the depot has available items and the rover has enough capacity.

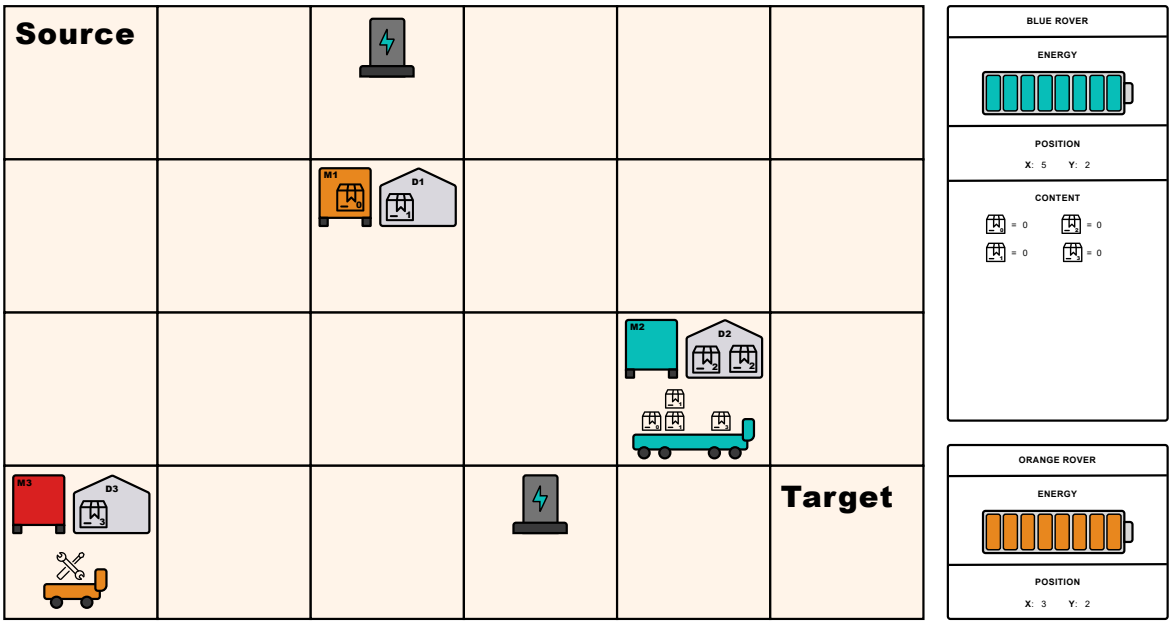
This model is characterized by 4 discrete variable that track the number of workitems of each type currently carried by the rover:
- `disc int[0..2] wi0`: Number of workitems of type 0 carried by the rover
- `disc int[0..2] wi1`: Number of workitems of type 1 carried by the rover.
- `disc int[0..2] wi2`: Number of workitems of type
- `disc int[0..2] wi3`: Number of workitems of type 3 carried by the rover.

This rover is needed to be on the same cell fom the machine to start processing a workitem using the event `MX_start`.

The rover can deliver finished workitems to the target using the `BR_deliver_3` events when it is on the same cell.

**Orange Rover (OR)**
The Orange Rover (OR) is responsible for repairing broken machines. It can only repair a machine when it is at the same coordinates as the machine with the `MX_repaired` event.

# Graphic Interface
This project contains a dynamic graphical interface that allows the user to visualize the state of the plant during the simulation.



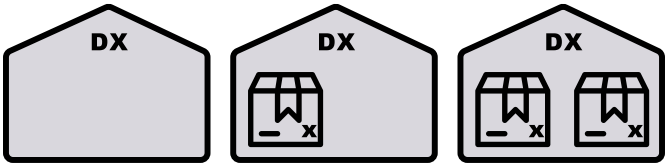

Each element in the graphic interface is dynamically updated based on the current state of the plant components.

## Machines


Each machine is represented by a rectangle that changes color based on its state:

- **Idle**: Light Blue
- **Working**: Orange
- **Broken**: Red

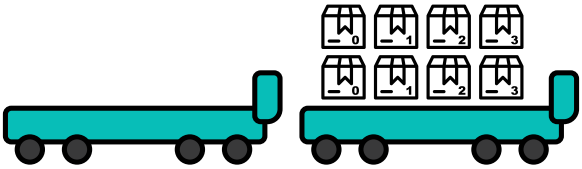
## Depots




Each depot represent the number of workitems stored using small icons inside the rectangle:

- **Empty Depot**: No icons
- **Partially Full Depot**: One icons
- **Full Depot**: Two icons

## Blue Rover



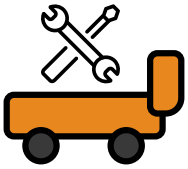
The Blue Rover is represented by a blue rover icon that moves within the plant according to its coordinates using the following code:

```
svgout id "B-Rover" attr "transform" value
  fmt("translate(%d, %d)", 52 + 207 * (BlueRover.x - 1), 135 + 207 * (4 - BlueRover.y))
```

All the *magic number* present are used to correctly position the rover inside the plant grid.

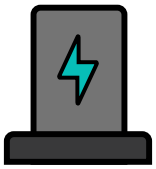
Also all the workitems currently carried by the rover are represented by small icons on top of the rover icon, with a maximum of 2 icons for each type of workitem.

## Orange Rover



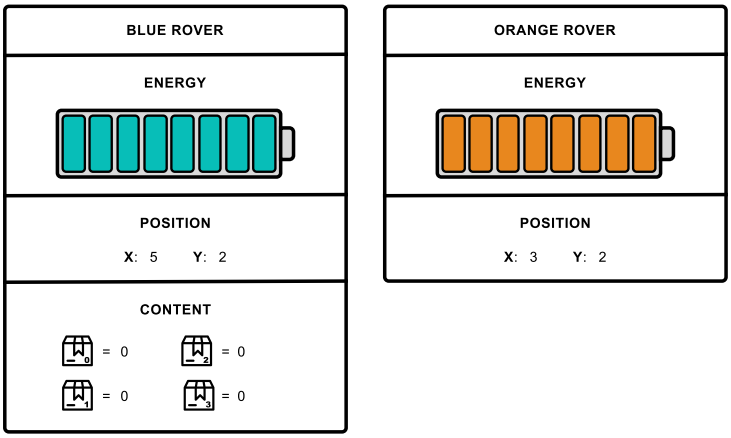
The Orange Rover is represented by an orange rover icon that moves within the plant according to its coordinates using the same code used for the Blue Rover, but with different magic numbers to correctly position the rover inside the plant grid.

## Charging Stations



The Charging Stations are represented by a gray charging station icon. They have no real state and are only used as reference points for the rovers to recharge their batteries.

### Rover User Interface




The user interface for the rovers is displayed in the right side of the plant and shows important information about the rovers' status:

- **Energy Level**: A simple battery that indicates the current energy level of the rover, with a maximum value of 8.
- **Position**: The current coordinates of the rover within the plant grid.
- **Carried Workitems**: A visual representation of the workitems currently carried by the Blue Rover, with a maximum of 2 icons for each type of workitem.

The user interface is dynamically updated based on the current state of the rovers, providing real-time feedback to the user during the simulation using the following code:

```
// Position Update
svgout id "BR-X-Pos" text value BlueRover.x;
svgout id "BR-Y-Pos" text value BlueRover.y;

// Energy Update
svgout id "BR-L1" attr "display" value if BlueRover.energy >= 1 : "inline" else "none" end;
svgout id "BR-L2" attr "display" value if BlueRover.energy >= 2 : "inline" else "none" end;
...
svgout id "BR-L8" attr "display" value if BlueRover.energy = 8 : "inline" else "none" end;

// Workitems Update
svgout id "BR-0-Label" text value BlueRover.wi0;
...
svgout id "BR-3-Label" text value BlueRover.wi3;
```

## Requirements

In this section are listed and explained the control policies that the supervisory controller must enforce to ensure the correct behavior of the plant.

### Requirement 1

*"No rover runs out of battery on tiles that are not charging stations"*

Battery preservation is a critical requirement for the rovers operating within the plant. If a rover depletes its battery while on a tile that is not a charging station, it would become immobilized. For this reason this control policy is enforced at plant level.

Let's consider the *up movement* of the Blue Rover as an example.

```
edge BR_move_up
  when y <= 3 and energy >= 1 and validMove(x, y + 1, energy - 1)
  do y := y + 1, energy := energy- 1;
```

Looking at the guard the movement is only allowed not oly if the rover stays within the plant boundaries (`y <= 3`) and has enough energy to perform the movement (`energy >= 1`), but also if the resulting position after the movement is valid using the `validMove`. This function simply check it the Manhattan distance from at least one of the two charging stations after the move is less than or equal to the remaining energy. The implementation is listed below and can be found on the `functions.cif` file.

```
func int[0..10] distance(int[0..7] x1; int[0..5] y1; int[1..6] x2; int[1..4] y2):
    return abs(x1 - x2) + abs(y1 - y2);
end

func bool validMove(int[0..7] x; int[0..5] y; int[-1..7] energy):
    return
      distance(x, y, Charger1_x, Charger1_y) <= energy or
      distance(x, y, Charger2_x, Charger2_y) <= energy;
end
```

## Requirement 2

*"The sums of the units of energy of the two batteries is always >= 1"*

At any moment the combined energy levels of the two rovers must be at least 1 unit. In order to prevent a situation where both rovers are completely out of energy, should check the resulting sum of their energy levels before allowing any movement.

```
requirement R2:
    location: initial; marked;
        edge BR_move_up, BR_move_down, BR_move_left, BR_move_right,
             OR_move_up, OR_move_down, OR_move_left, OR_move_right
             when BlueRover.energy + OrangeRover.energy - 1 >= 1;
end
```

The guard `BlueRover.energy + OrangeRover.energy - 1 >= 1` ensures that **after** any movement event, the total energy of both rovers remains at least 1 unit.

## Requirement 3

*"The maximum number of workpieces that the blue rover can carry is 4"*

At any moment the Blue Rover can carry at most 4 workitems in total, regardless of their type. This control policy is enforced using a requirement that checks the sum of all workitems currently carried by the rover before allowing it to pick up another workitem from a depot.

```
requirement R3:
    location: initial; marked;
```

```
        edge BR_take_0, BR_take_1, BR_take_2, BR_take_3
            when BlueRover.wi0 + BlueRover.wi1 + BlueRover.wi2 + BlueRover.wi3 < 4;
end
```

## Requirement 4

*"Each rover can charge provided its battery is not already full"*

To prevent unnecessary charging actions, each rover is allawed to charge only whel its battery is not already full. This control policy is enforced using a requirement invariant that checks the current energy level of the rover before allowing it to perform the charging action.

```
requirement invariant R4A: BR_charge needs BlueRover.energy < 8;
requirement invariant R4B: OR_charge needs OrangeRover.energy < 8;
```

## Requirement 5

*"Rovers do not collide"*

Two rovers must never occupy the same position within the plant at the same time. This control policy is enforced using a requirement that checks if the two rovers are on adiacent tiles, if so the movement that would cause a collision is prevented.

```
requirement R5:
    location: initial; marked;
        edge BR_move_up, OR_move_down
          when (BlueRover.y + 1 != OrangeRover.y and BlueRover.x = OrangeRover.x) or
               OrangeRover.x != BlueRover.x;

        edge BR_move_right, OR_move_left
          when (BlueRover.x + 1 != OrangeRover.x and BlueRover.y = OrangeRover.y) or
               OrangeRover.y != BlueRover.y;

        edge BR_move_down, OR_move_up
          when (BlueRover.y - 1 != OrangeRover.y and BlueRover.x = OrangeRover.x) or
               OrangeRover.x != BlueRover.x;

        edge BR_move_left, OR_move_right
          when (BlueRover.x - 1 != OrangeRover.x and BlueRover.y = OrangeRover.y) or
               OrangeRover.y != BlueRover.y;
end
```

## Requirement 6

*"If D3 is full, then D1 can store at most one workpiece"*

If Depot D3 contains 2 workpieces, then Depot D1 is only allowed to store at most one workitem. This control policy is enforced using a requirement invariant that checks the quantity of workitems in both depots before allowing Machine M1 to start another process. This is necessary since `M1_start` event is controllable while the `M3_finish_success` event is uncontrollable.

```
requirement invariant R6: M1_start needs D1.quantity < 1 or D3.quantity < 2;
```

## Tooldef

The final stage of the project involved the automation of the supervisor generation process using a `tooldef` file that specifies the necessary configurations.

The `tooldef` file is really simple and is composed by three main section:

1. **Enviroment Checking**: If the `generated` directory does not exist, it is created to store the generated CIF files.
2. **Supervisor Synthesis**: The `cifdatasynth` command is used to generate the supervisor based on the plant requirements specified in the `plant/requirements.cif` file. The output is saved in the `generated/supervisor.cif` file.
3. **Plant Merging**: The `cifmerge` command is used to combine the original plant model with the synthesized supervisor, resulting in a controlled system saved in the `generated/controlled-system.cif` file.

```
from "lib:cif" import *;

if not exists("generated"):
    mkdir("generated");
end

cifdatasynth(
    "plant/requirements.cif",
    "-o generated/supervisor.cif",
    "-n supervisor"
);

cifmerge(
    "plant/plant.cif",
    "generated/supervisor.cif",
     "-o generated/controlled-system.cif"
);
```

## Conclusions