

**CAMUNDA  
CON  
LIVE**

# Controlling a Smart Factory with Processes in Camunda

Ronny Seiger, University of St. Gallen





University of St.Gallen



Universität Trier



German  
Research Center  
for Artificial  
Intelligence

## Controlling a Smart Factory with Processes in Camunda

Ronny Seiger, Barbara Weber  
University of St.Gallen, Switzerland

Lukas Malburg, Ralph Bergmann  
University of Trier, Germany  
German Research Center for AI (DFKI)



CamundaCon Live 2021  
22 September 2021

*"From insight  
to impact"* 

## Ronny Seiger

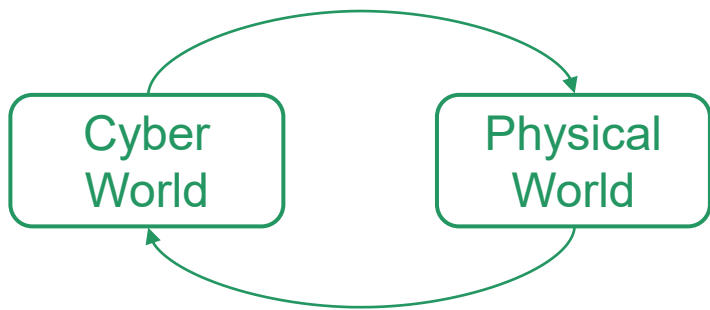
- Diploma (2011) & PhD (2018) in Computer Science at TU Dresden
- Post-Doc at University of St.Gallen since 2019
- Research
  - Cyber-physical Systems, Internet of Things
  - Software Engineering, Software Architecture
  - Business Process Management
- Teaching
  - BPM, Software Engineering

[ronny.seiger@unisg.ch](mailto:ronny.seiger@unisg.ch)

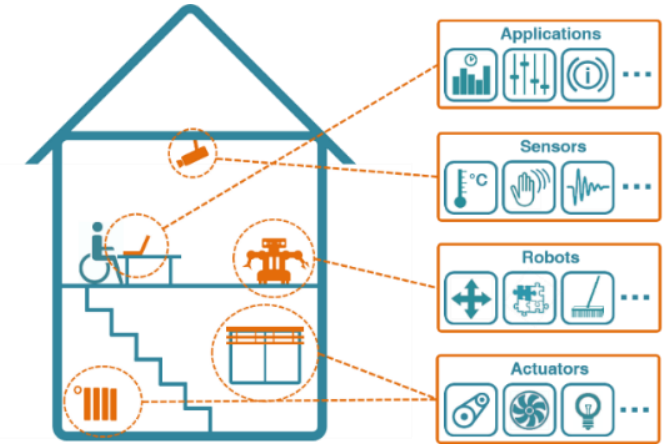


# Emergence of Smart Spaces

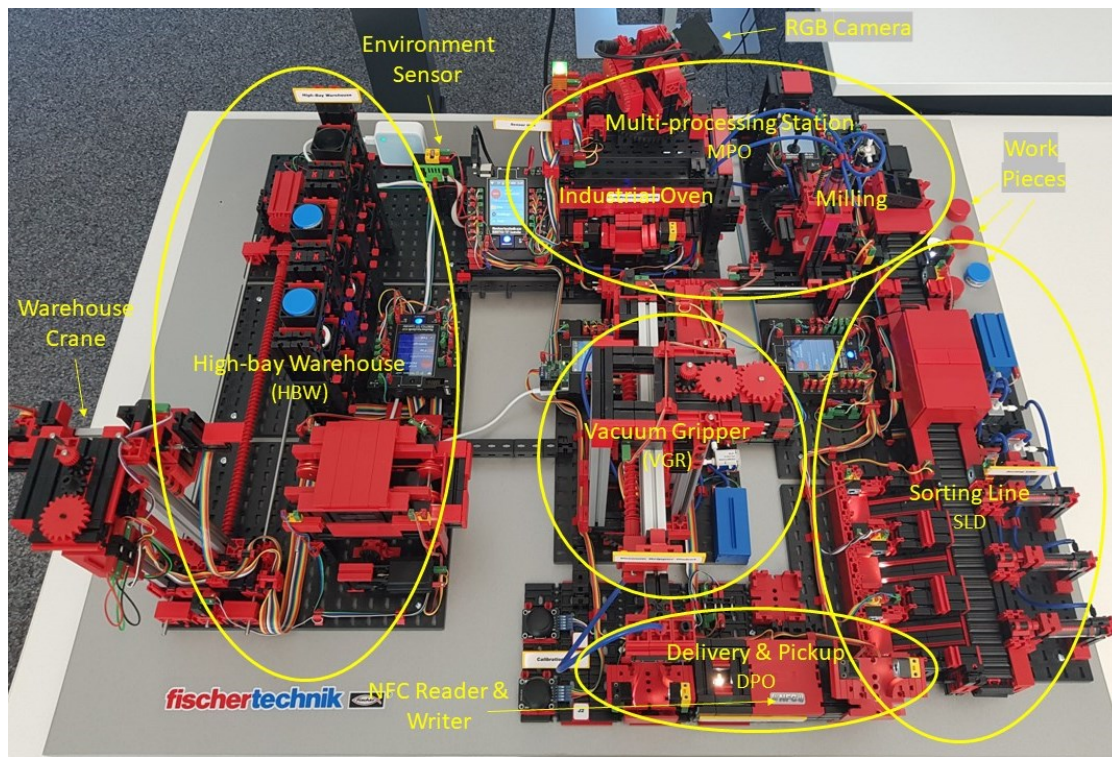
Cyber-physical Convergence



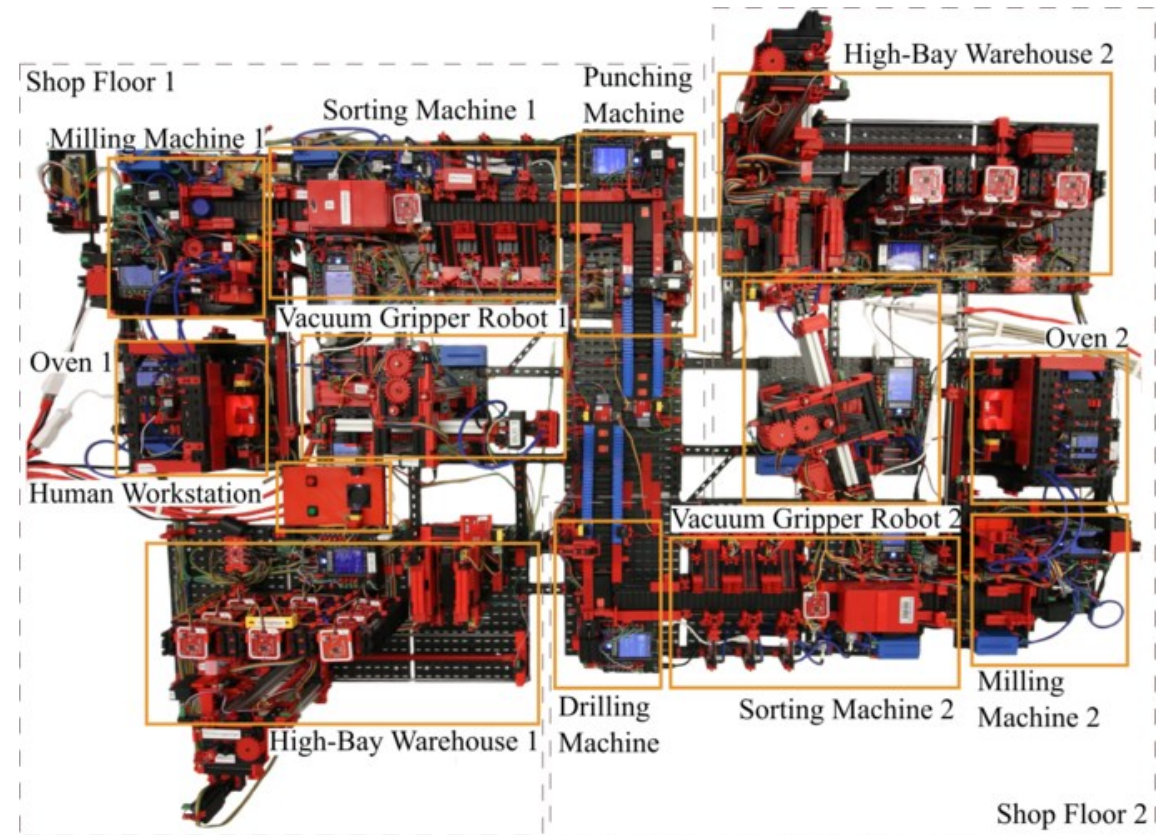
Cyber-physical System



# Smart Factory Simulation Models

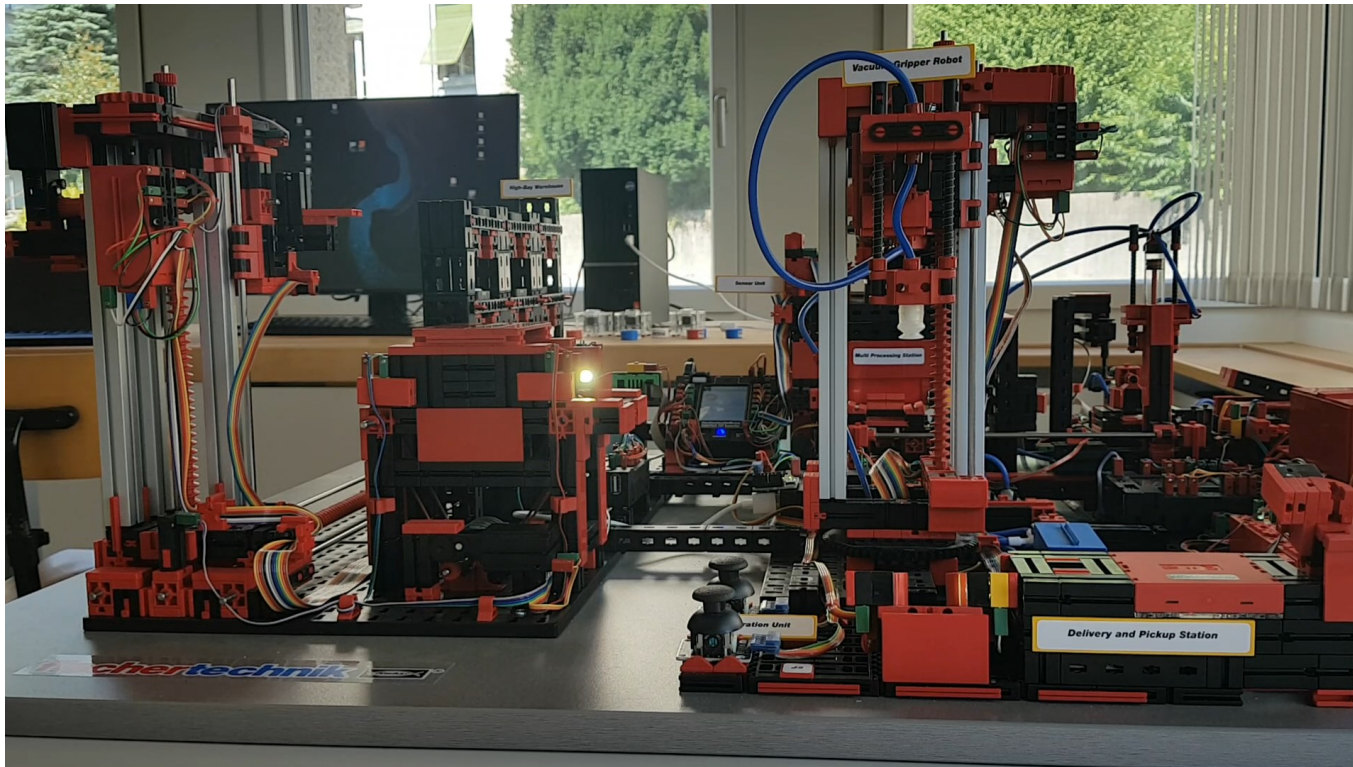
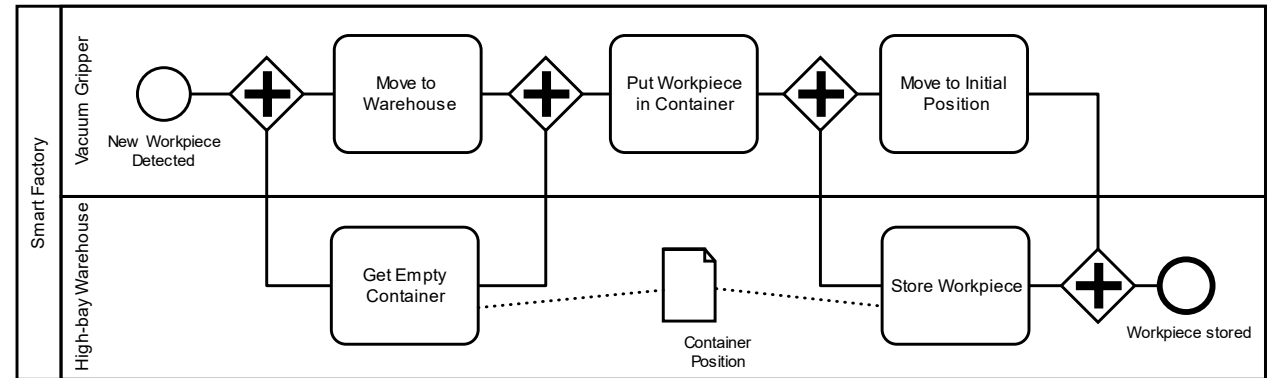


Uni St.Gallen

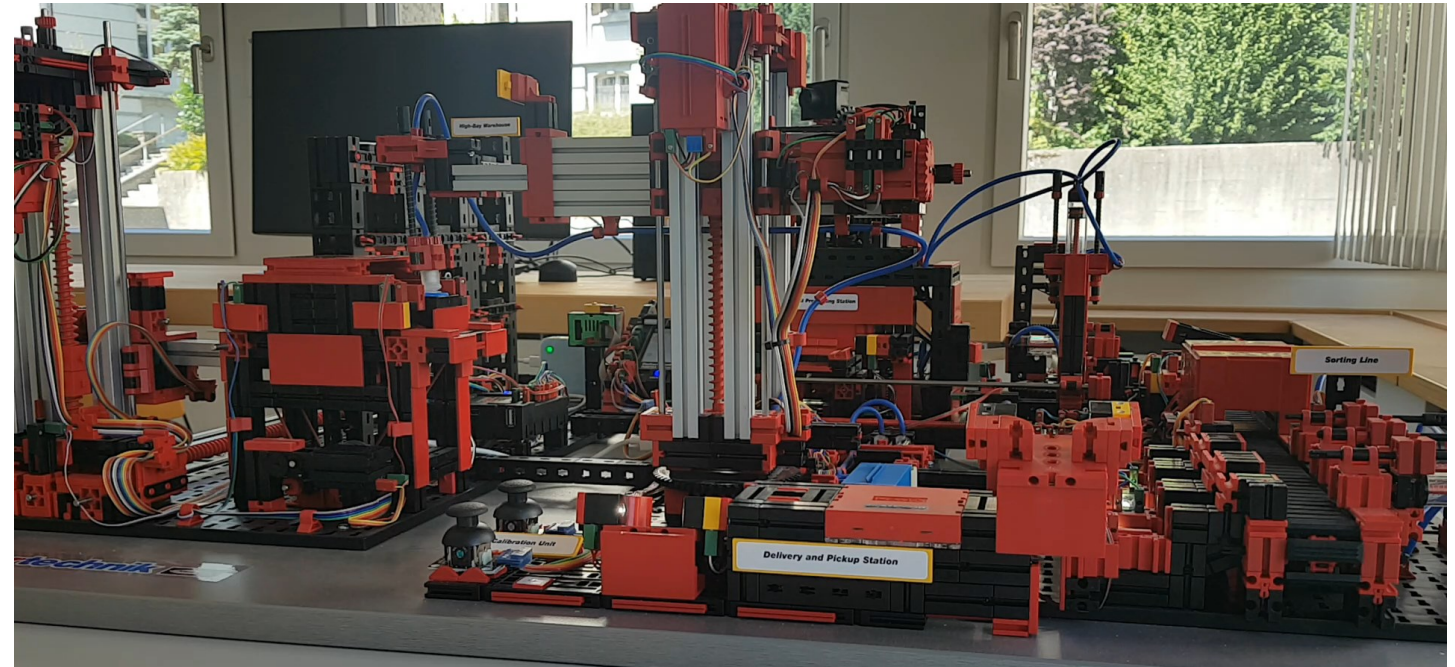
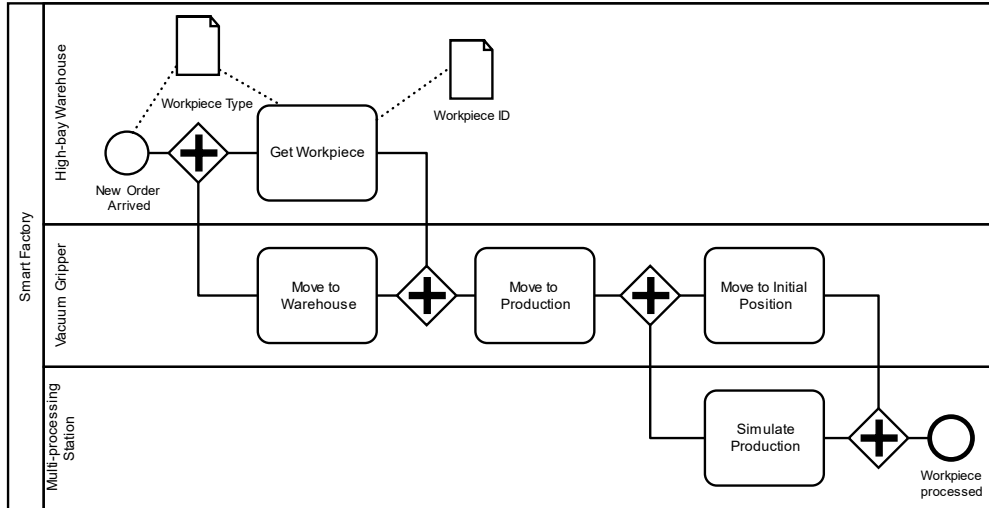


Uni Trier

## Smart Factory Process: Storage



# Smart Factory Process: Production

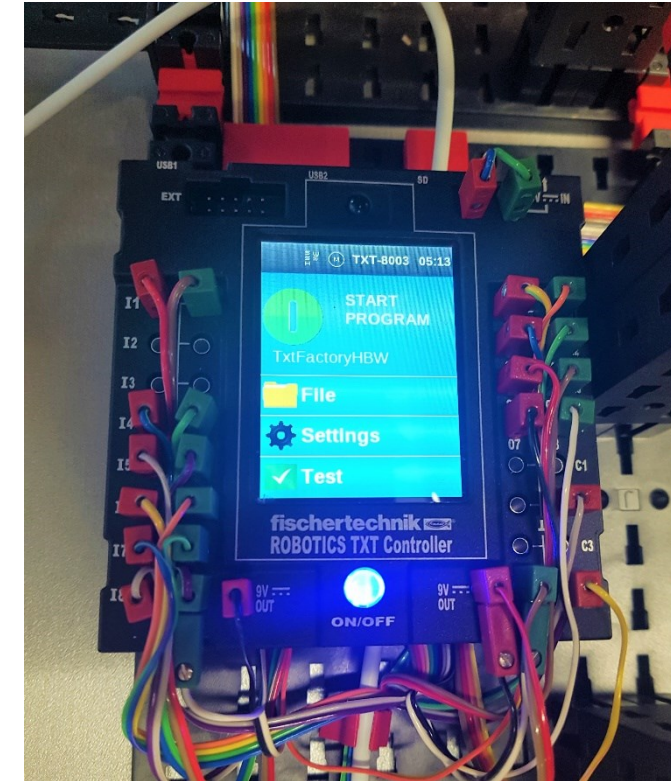


## Programming the Factories: As-is State

```

#elif CLIENT_HBW
  if (msg->get_topic() == TOPIC_OUTPUT_STATE_ACK) {
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "DETECTED state ack:{}", msg->get_topic());
    std::stringstream ssin(msg->to_string());
    Json::Value root;
    try {
      ssin >> root;
      std::string sts = root["ts"].asString();
      SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " ts:{}", sts);
      if (ft::trycheckTimestampTTL(sts))
      {
        hbw_.requestQuit();
      }
    } catch (const Json::RuntimeError& exc) {
      std::cout << "Error: " << exc.what() << std::endl;
    }
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "OK.", 0);
  } else if (msg->get_topic() == TOPIC_LOCAL_SSC_JOY) {
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "DETECTED joy:{}", msg->get_topic());
    std::stringstream ssin(msg->to_string());
    Json::Value root;
    try {
      ssin >> root;
      std::string sts = root["ts"].asString();
      SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " ts:{}", sts);
      if (ft::trycheckTimestampTTL(sts))
      {
        ft::TxtJoysticksData jd;
        jd.aX1 = root["aX1"].asInt();
        jd.aY1 = root["aY1"].asInt();
        jd.b1 = root["b1"].asBool();
        jd.aX2 = root["aX2"].asInt();
        jd.aY2 = root["aY2"].asInt();
        jd.b2 = root["b2"].asBool();
        hbw_.requestJoyBut(jd);
        SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " 1: {} {} {} 2: {} {} {}", jd.aX1, jd.aY1, jd.b1, jd.aX2,
        } catch (const Json::RuntimeError& exc) {
          std::cout << "Error: " << exc.what() << std::endl;
        }
        SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "OK.", 0);
      } else if (msg->get_topic() == TOPIC_LOCAL_VGR_DO) {
        SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "DETECTED vgr do:{}", msg->get_topic());

```



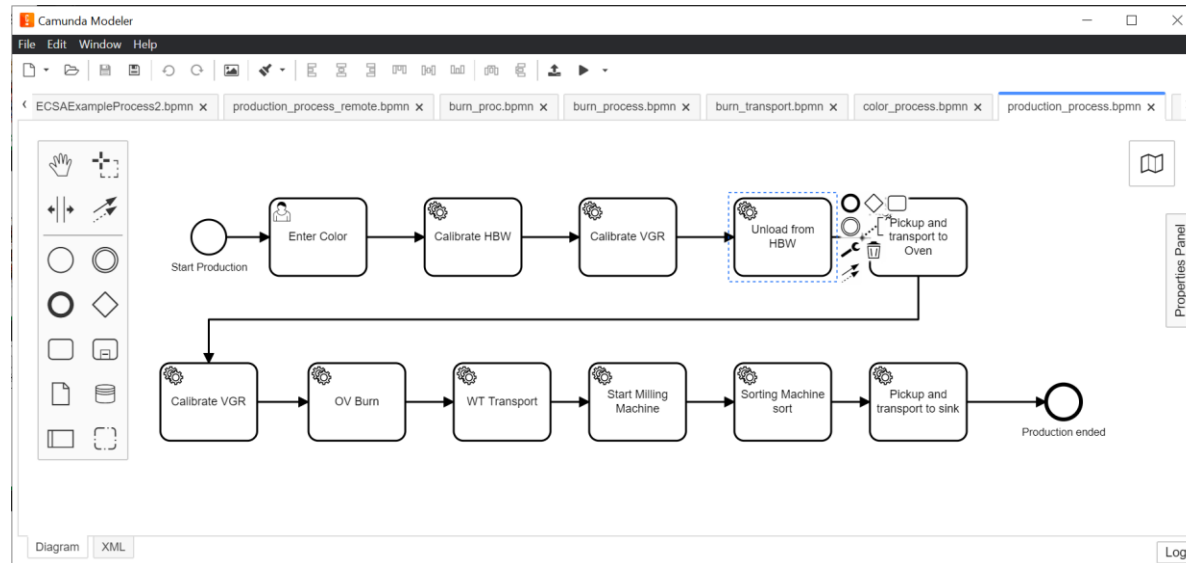
6–12 Embedded TXT Controllers

- 1 C++ application / controller
- 2 hard-wired processes

[https://github.com/fischertechnik/txt\\_training\\_factory](https://github.com/fischertechnik/txt_training_factory)

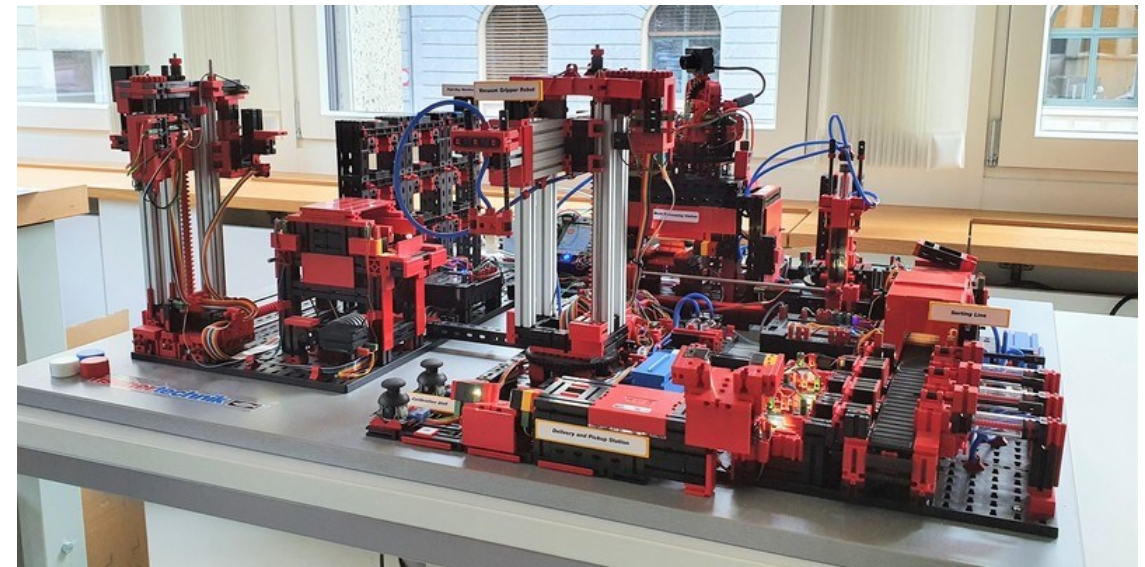


## «Programming» the Factories: To-be State



Model Process in BPMN in Camunda Modeler (*low-code*)

Execute via Camunda Platform

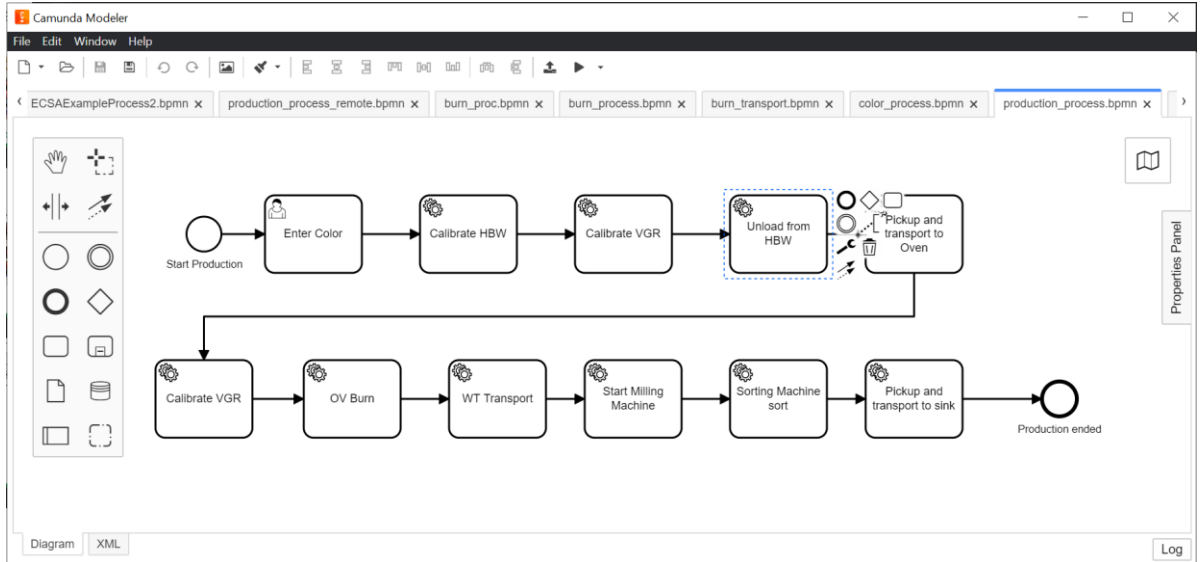
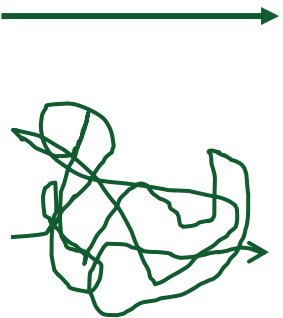


# The Journey

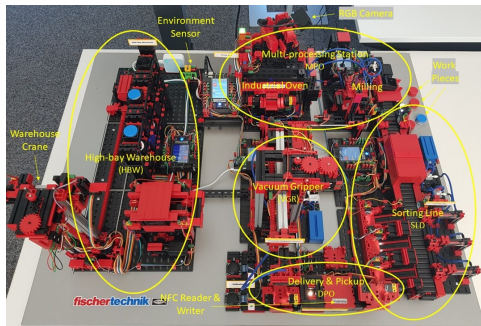
```

#elif CLIENT_HBW
if (msg->get_topic() == TOPIC_OUTPUT_STATE_ACK) {
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "DETECTED state ack:{}", msg->get_topic());
    std::stringstream ssin(msg->to_string());
    Json::Value root;
    try {
        ssin >> root;
        std::string sts = root["ts"].asString();
        SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " ts:{}", sts);
        if (ft::trycheckTimestampTTL(sts))
        {
            hbw_.requestQuit();
        }
    } catch (const Json::RuntimeError& exc) {
        std::cout << "Error: " << exc.what() << std::endl;
    }
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "OK.", 0);
} else if (msg->get_topic() == TOPIC_LOCAL_SSC_JOY) {
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "DETECTED joy:{}", msg->get_topic());
    std::stringstream ssin(msg->to_string());
    Json::Value root;
    try {
        ssin >> root;
        std::string sts = root["ts"].asString();
        SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " ts:{}", sts);
        if (ft::trycheckTimestampTTL(sts))
        {
            ft::TxtJoysticksData jd;
            jd.ax1 = root["ax1"].asInt();
            jd.av1 = root["av1"].asInt();
            jd.b1 = root["b1"].asBool();
            jd.ax2 = root["ax2"].asInt();
            jd.av2 = root["av2"].asInt();
            jd.b2 = root["b2"].asBool();
            hbw_.requestJoyBut(jd);
            SPDLOG_LOGGER_DEBUG(spdlog::get("console"), " 1:{} {} 2:{} {} {} ", jd.ax1, jd.av1, jd.b1, jd.ax2
        }
    } catch (const Json::RuntimeError& exc) {
        std::cout << "Error: " << exc.what() << std::endl;
    }
    SPDLOG_LOGGER_DEBUG(spdlog::get("console"), "OK.", 0);
} else if (msg->get_topic() == TOPIC_LOCAL_VGR_DO) {

```

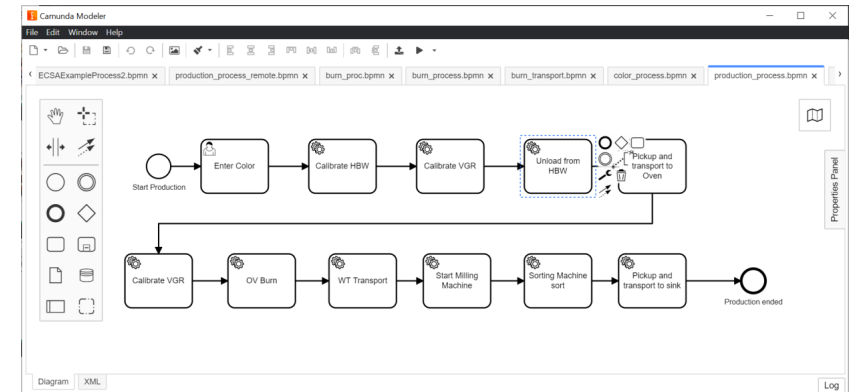


# Agenda

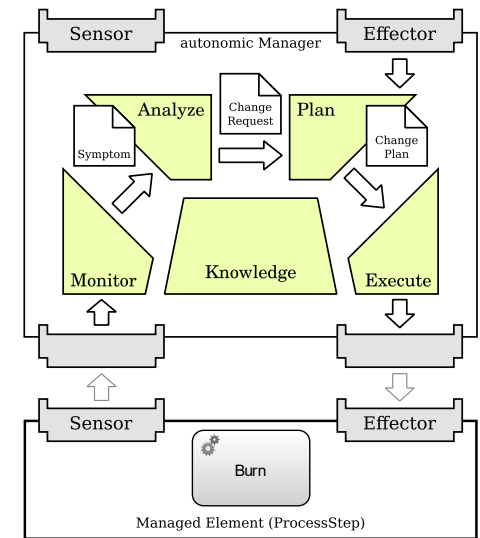
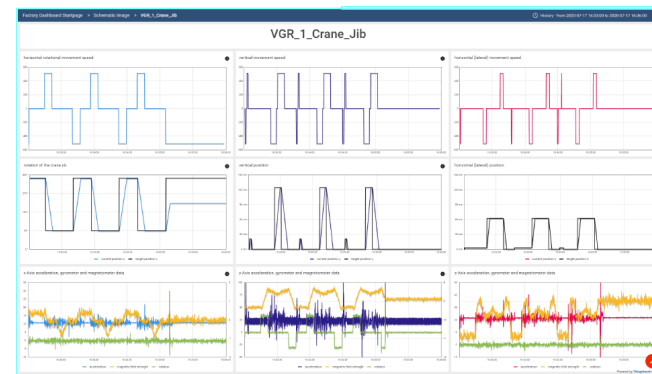


```
if (msg-get_topic() == TOPIC_OUTPUT_START_ACK) {
  SPOLOG_LOGGER_DEBUG(splog::get("console"), "DETECTED state ack()", msg-get_topic());
  std::stringstream ssin(msg->str_string());
  json::value root;
  try {
    ssin >> root;
    std::string sts = root["st"].asString();
    SPOLOG_LOGGER_DEBUG(splog::get("console"), " " + sts);
    if (!root["request"].isString()) {
      HbW_requestQuit();
    }
  } catch (const json::RuntimeError& ex) {
    std::cout << "Error: " << ex.what() << std::endl;
  }
  SPOLOG_LOGGER_DEBUG(splog::get("console"), "OK", 0);
} else if (msg-get_topic() == TOPIC_LOCAL_SOC_DONE) {
  SPOLOG_LOGGER_DEBUG(splog::get("console"), "DETECTED socy()", msg-get_topic());
  std::stringstream ssin(msg->str_string());
  json::value root;
  try {
    ssin >> root;
    std::string sts = root["st"].asString();
    SPOLOG_LOGGER_DEBUG(splog::get("console"), " " + sts);
    if (!root["request"].isString()) {
      ft_TrajectoryData Soc;
      Soc.ak1 = root["ak1"].asInt();
      Soc.ak2 = root["ak2"].asInt();
      Soc.bs1 = root["bs1"].asBool();
      Soc.bs2 = root["bs2"].asBool();
      Soc.ak3 = root["ak3"].asInt();
      Soc.ak4 = root["ak4"].asInt();
      Soc.bs3 = root["bs3"].asBool();
      Soc.bs4 = root["bs4"].asBool();
      HbW_requestTrajectory();
      SPOLOG_LOGGER_DEBUG(splog::get("console"), " 1:() () 2:() () 3:, 34-ak3, 34-ak4, 34-bs1, 34-bs2, 34-ak3, 34-ak4, 34-bs3, 34-bs4");
    }
  } catch (const json::RuntimeError& ex) {
    std::cout << "Error: " << ex.what() << std::endl;
  }
  SPOLOG_LOGGER_DEBUG(splog::get("console"), "OK", 0);
} else if (msg-get_topic() == TOPIC_LOCAL_VGR_DO) {

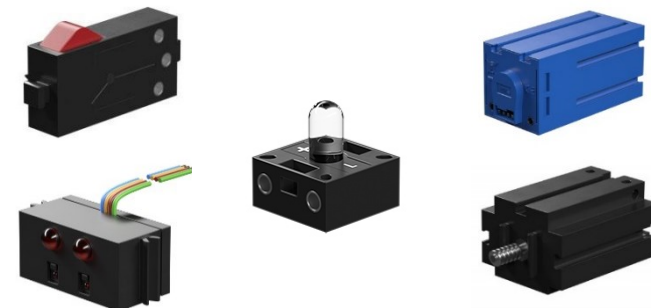
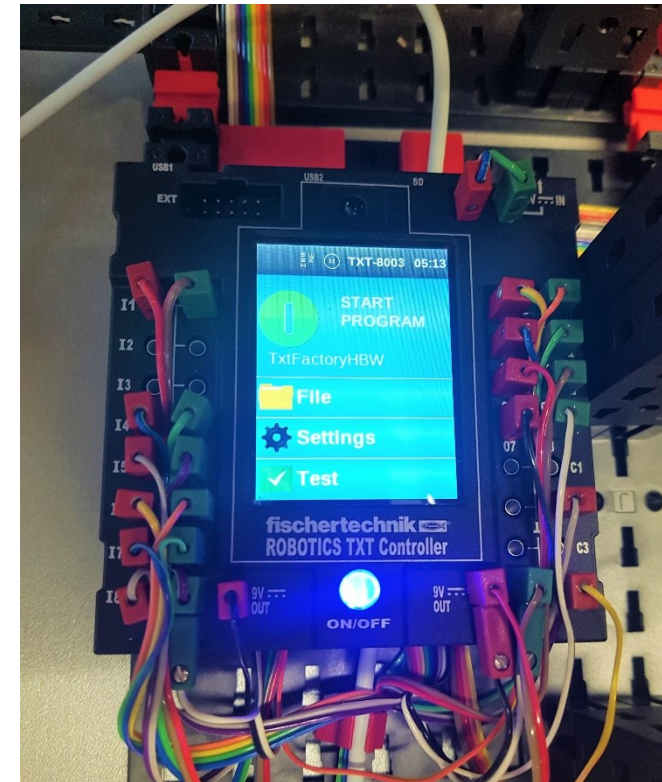
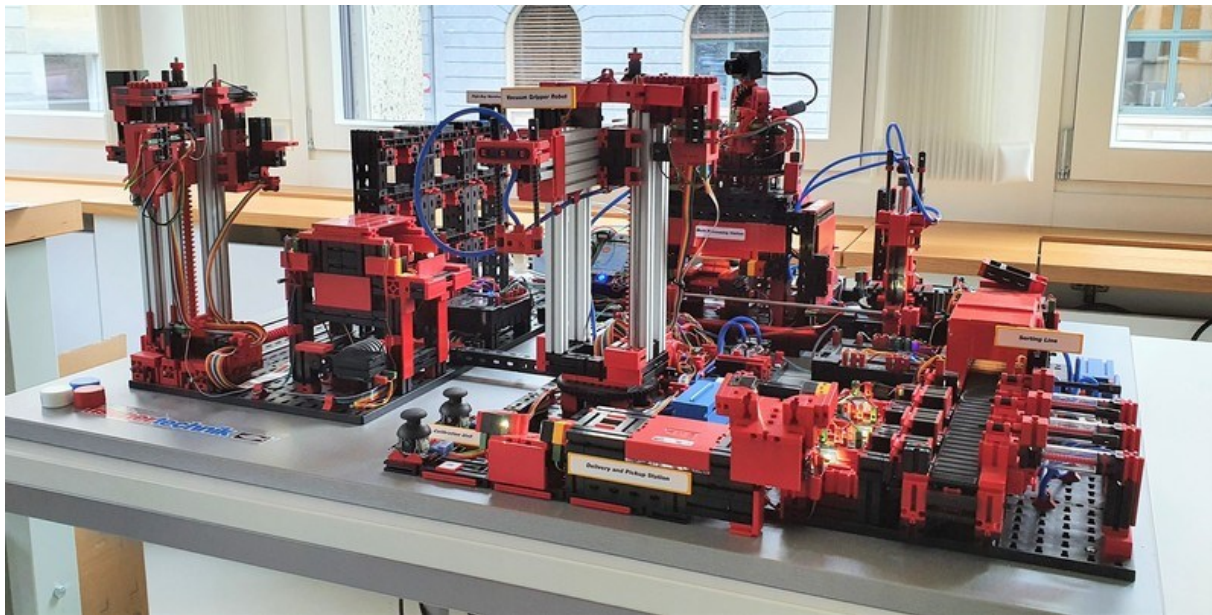
```



- Hardware
- Programming
  - Low-level & High-level
- Web Services & Business Processes
- Sensors
  - Stream Processing & Machine Learning
- Research: Autonomous Processes

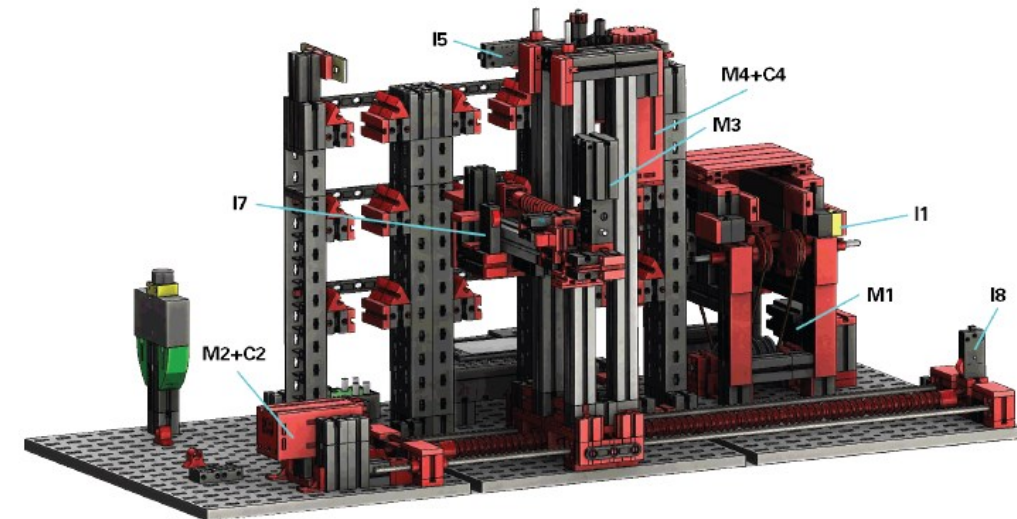


# The Embedded TXT Controllers: Hardware



## Know Your Wiring!

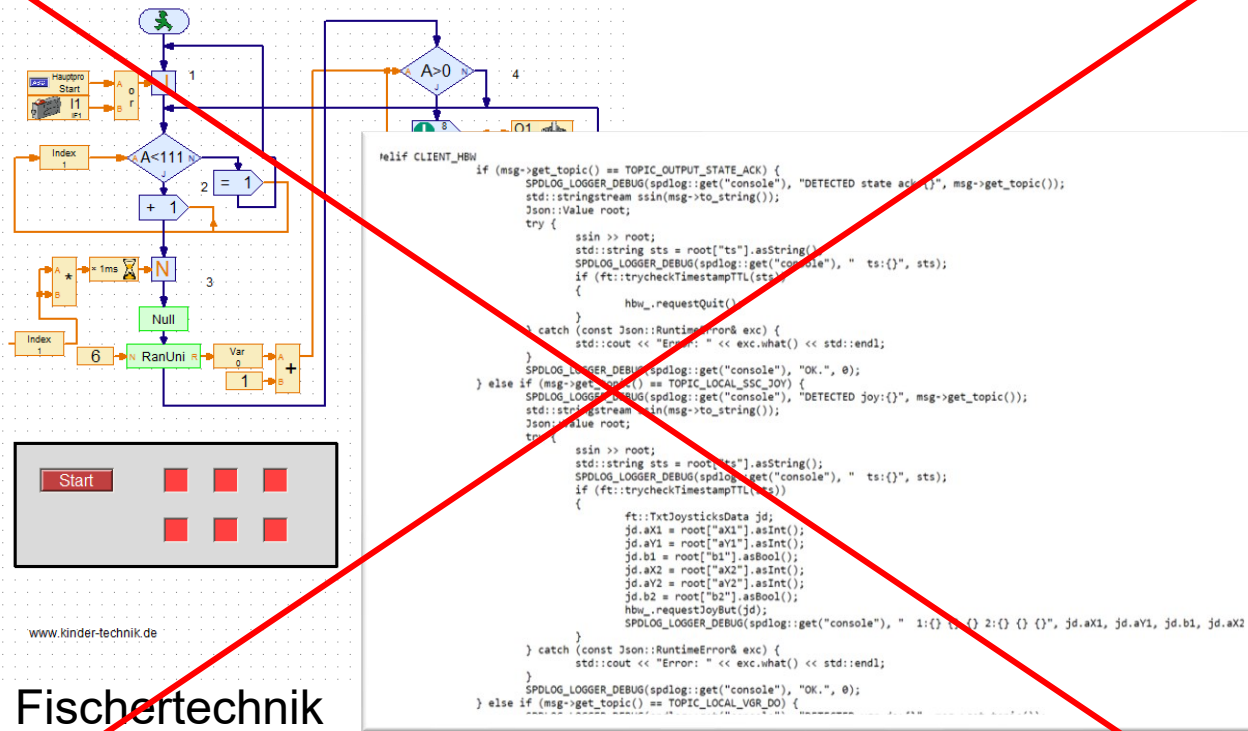
| Number | Function                             | Input / output |
|--------|--------------------------------------|----------------|
| 1      | Photo-transistor external            | I1             |
| 2      | Photo-transistor internal            | I4             |
| 3      | Reference pushbutton horizontal      | I5             |
| 4      | Reference pushbutton probe arm rear  | I6             |
| 5      | Reference pushbutton probe arm front | I7             |
| 6      | Reference pushbutton vertical        | I8             |
| 7      | Encoder horizontal                   | C2             |
| 8      | Encoder vertical                     | C4             |
| 9      | Motor conveyor belt                  | M1             |
| 10     | Horizontal motor                     | M2             |
| 11     | Motor probe arm                      | M3             |
| 12     | Vertical motor                       | M4             |



from [https://www.fischertechnik.de/-/media/fischertechnik/fite/service/elearning/lehren/lernfabrik/fabrik\\_2019\\_englisch\\_neu.ashx](https://www.fischertechnik.de/-/media/fischertechnik/fite/service/elearning/lehren/lernfabrik/fabrik_2019_englisch_neu.ashx)

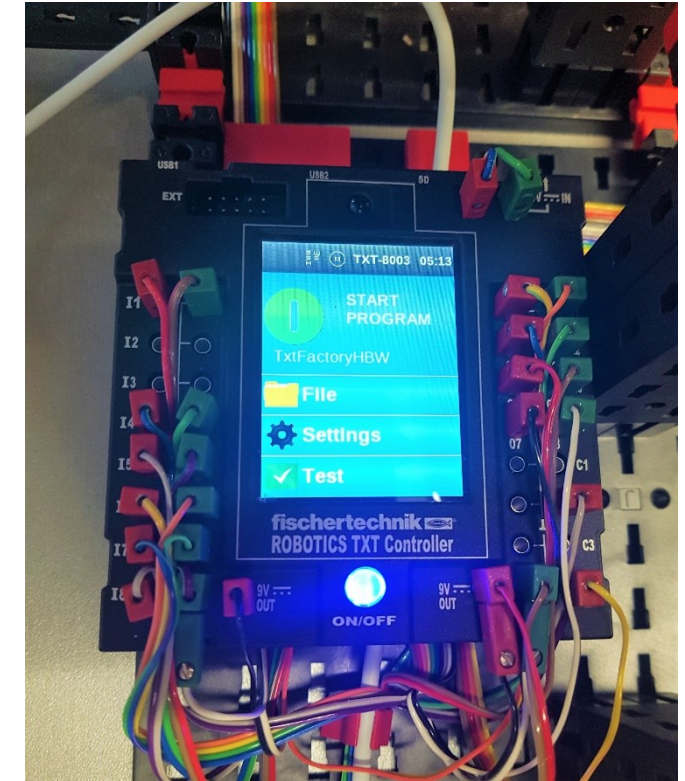
# Hardware ✓

# The Embedded TXT Controllers: Software



Fischertechnik RoboPro

Fischertechnik C++ Project



fischertechnik TXT community firmware

ftrobopy

<https://github.com/ftrobopy/ftrobopy>

## The Embedded TXT Controllers: Software

### ftrobopy

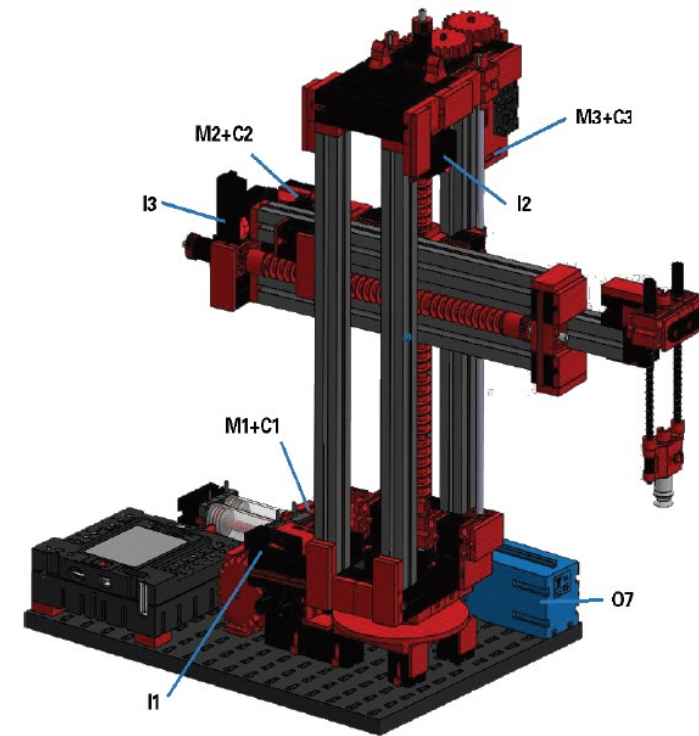
- Unix Process on controller
  - Provides TCP Sockets
- Python library
  - Deploy locally or on remote computer



```
1 import ftrobopy
2
3 my_txt1 = ftrobopy.ftrobopy(host=f"192.168.0.12", port=65000, update_interval=0.01)
4
```

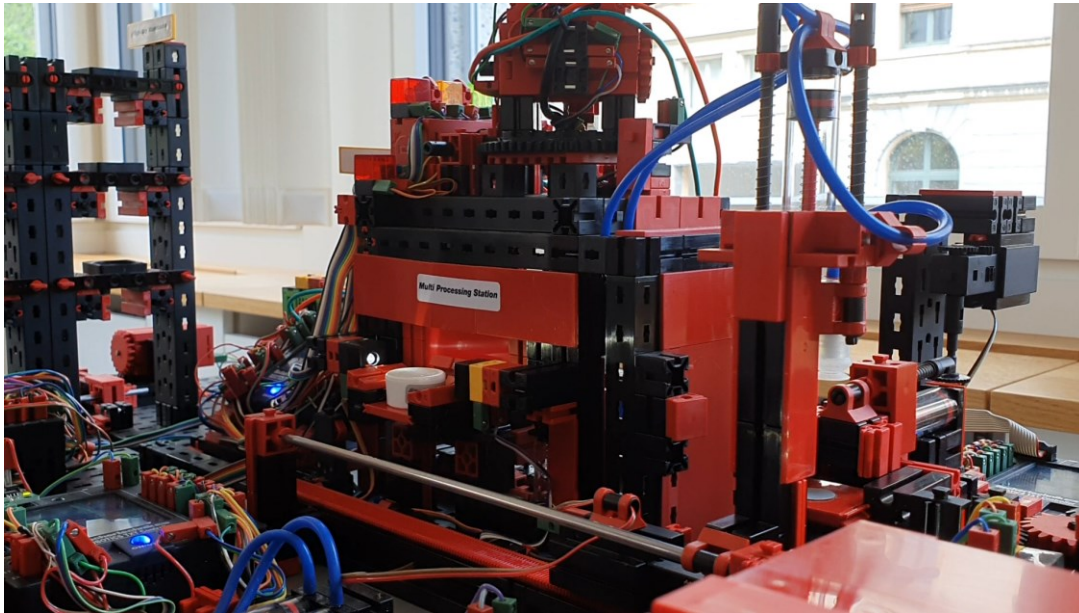
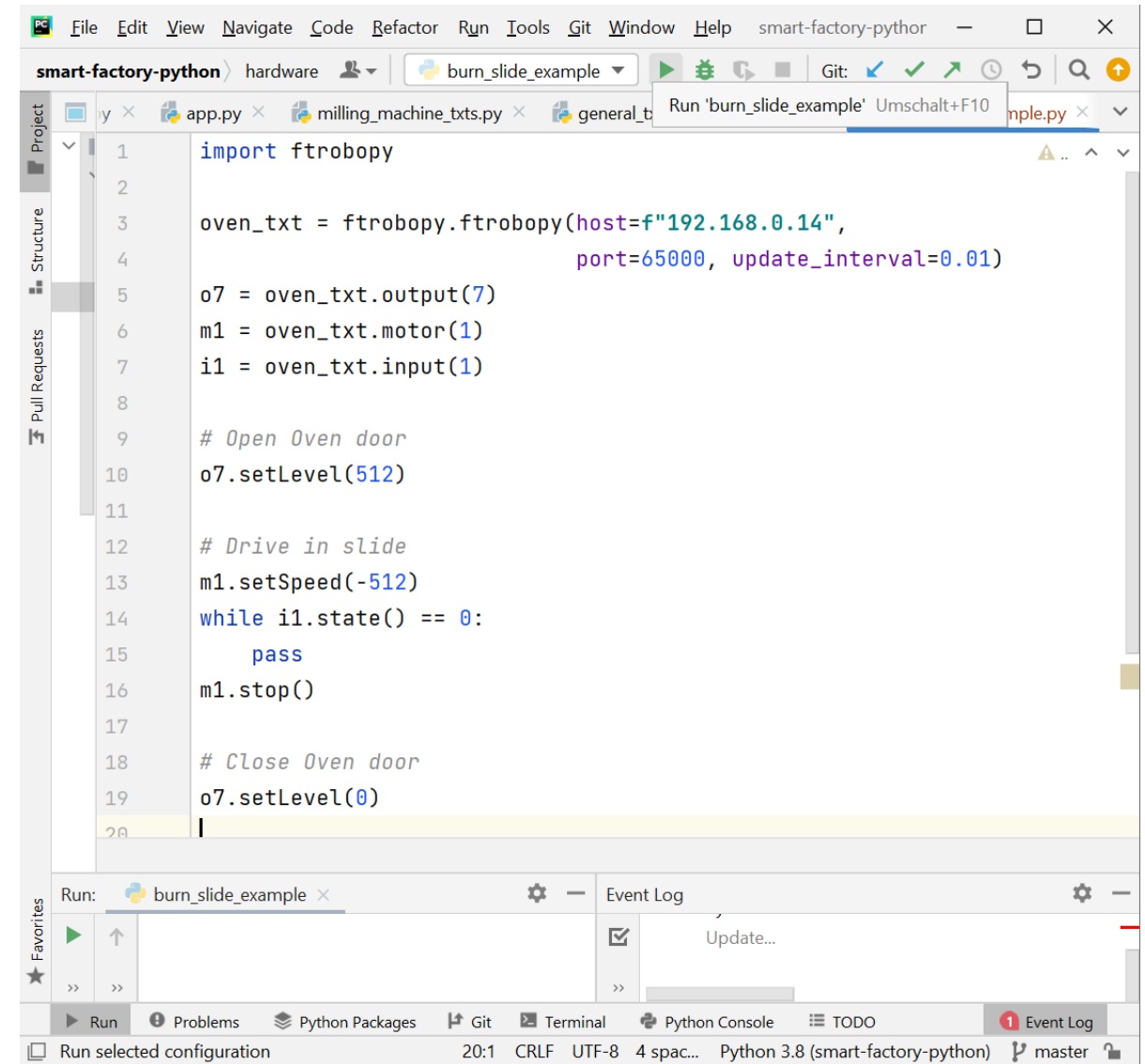
## Bringing Hardware & Software Together

```
5   m1 = my_txt1.motor(1)
6   m1.setSpeed(512)
7
8   m2 = my_txt1.motor(2)
9   m2.setSpeed(-256)
10
11  o7 = my_txt1.output(7)
12  o7.setLevel(512)
13
14  i3 = my_txt1.input(3)
15  if i3.state() == 1:
16      print("Switch i3 pressed")
```





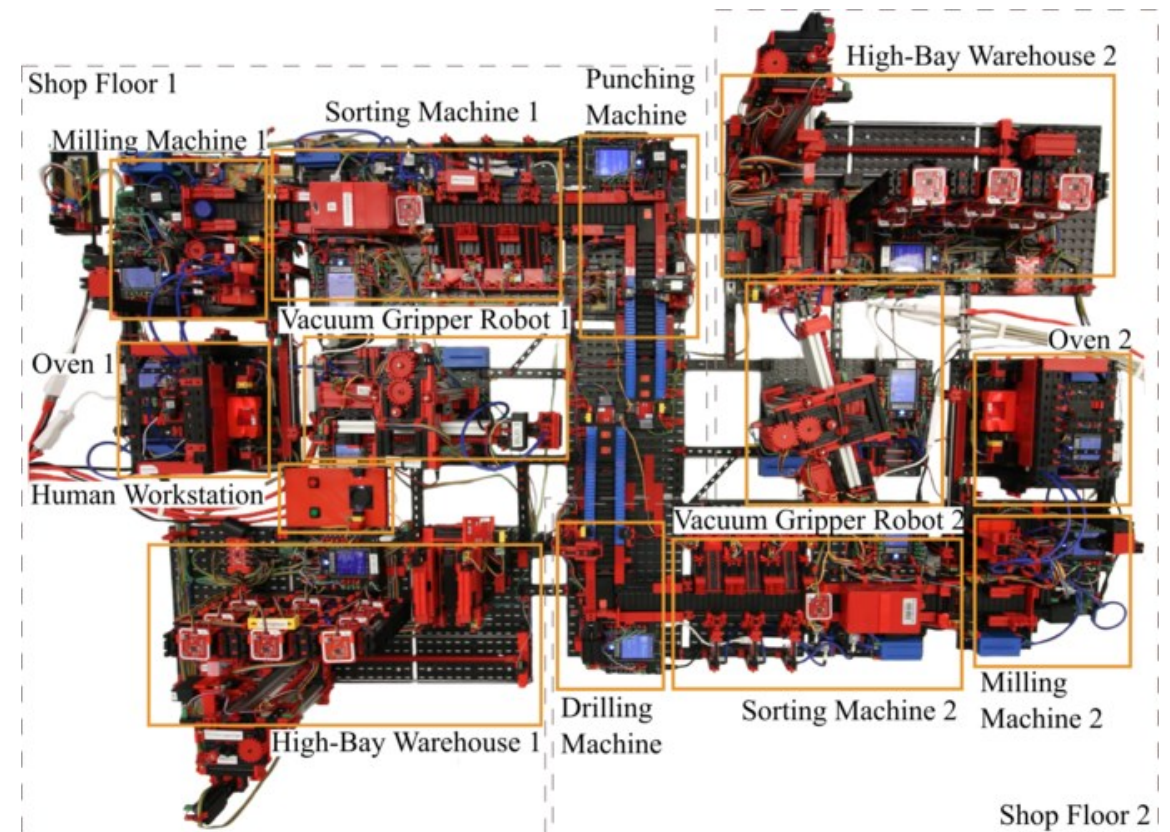
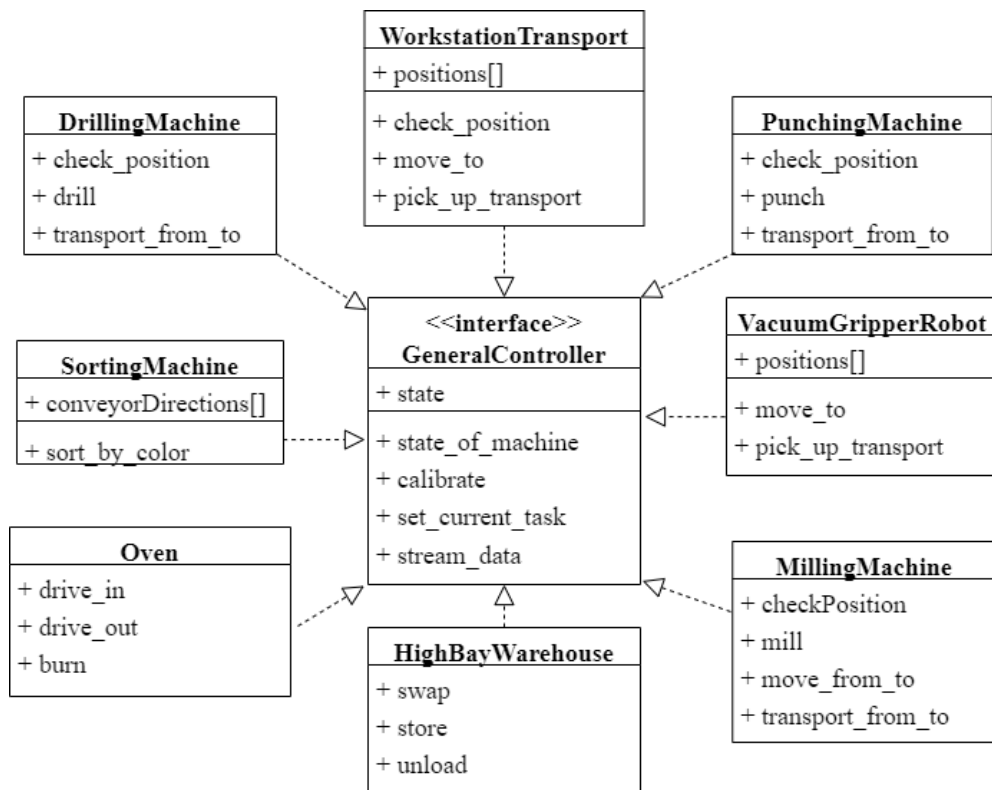
## Small Example: Oven

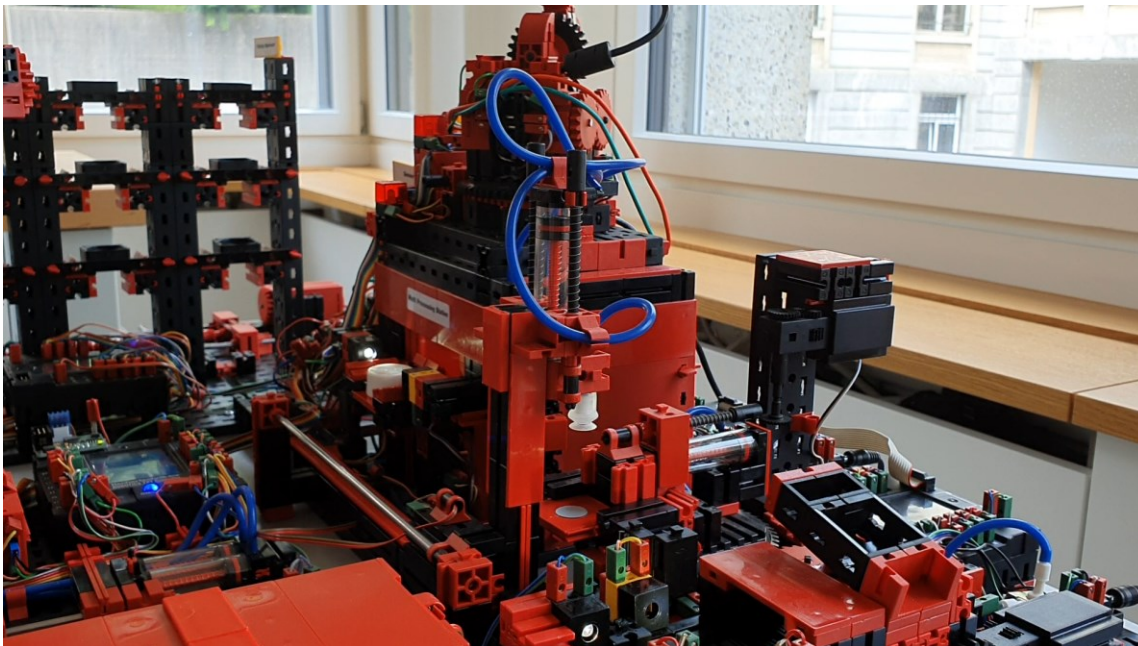
```

1  import ftrrobpy
2
3  oven_txt = ftrrobpy.ftrrobpy(host=f"192.168.0.14",
4                                     port=65000, update_interval=0.01)
5
6  o7 = oven_txt.output(7)
7  m1 = oven_txt.motor(1)
8  i1 = oven_txt.input(1)
9
10 # Open Oven door
11 o7.setLevel(512)
12
13 # Drive in slide
14 m1.setSpeed(-512)
15 while i1.state() == 0:
16     pass
17     m1.stop()
18
19 # Close Oven door
20 o7.setLevel(0)
  
```

# Going Object-oriented



## Going Object-oriented: Example



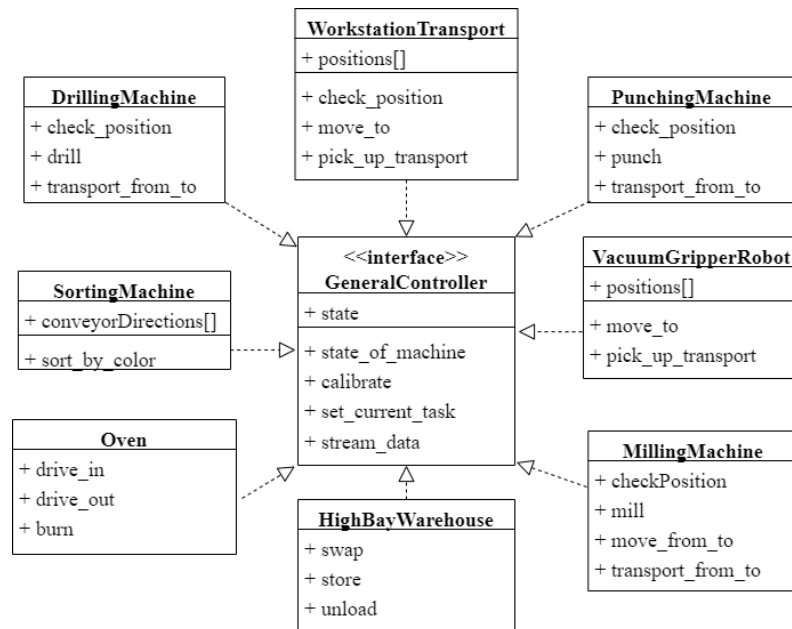
```

File Edit View Navigate Code Refactor Run Tools Git Window Help smart-factory-python - burn_oop.py
smart-factory-python > hardware > generic > burn_slide_example
ple.py x OV1.py x multi_processing_station_txts.py x ftrobopy.py x
Run 'burn_slide_example' Umschalt+F10

1 oven1 = Oven(12)
2 oven1.burn(5)
3
4 class Oven:
5     def __init__(self, txt_number):
6         ...
7
8     def burn(self, time_in_seconds: int = 2) -> None:
9         self._drive_into_oven()
10        self.txt.play_sound(15)
11        for i in range(time_in_seconds*8):
12            if i%2==0:
13                self.o8.setLevel(512)
14            else: self.o8.setLevel(0)
15                time.sleep(0.3)
16        self.o8.setLevel(0)
17        self._drive_out_of_oven()
18        self._adjust_platform_for_workstation_transport()
19        return
20        ...
21
22    def _adjust_platform_for_workstation_transport(self) -> None:
23        self.m1.setDistance(100)
24        self.m1.setSpeed(-1 * self.m1_speed)
25        time.sleep(0.3)
26        self.m1.stop()
27        return
  
```

## Going Service-based

- Expose high-level methods of OOP model 1:1 via RESTful web service
- HTTP GET with parameters
- Python Flask



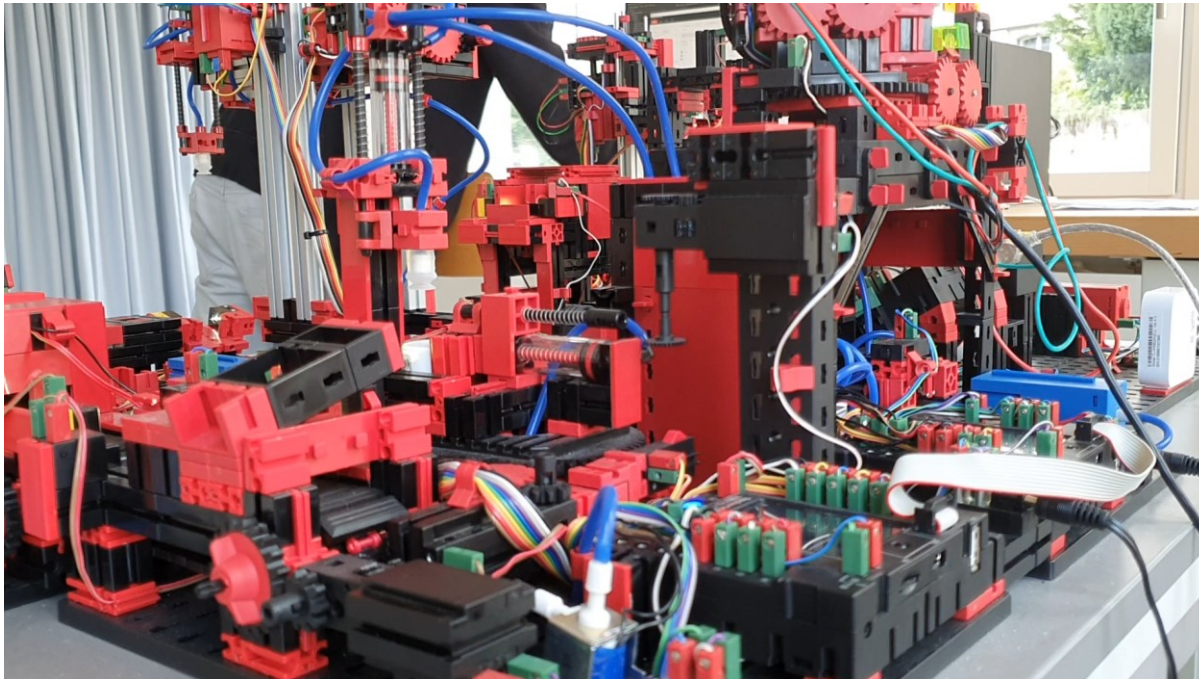
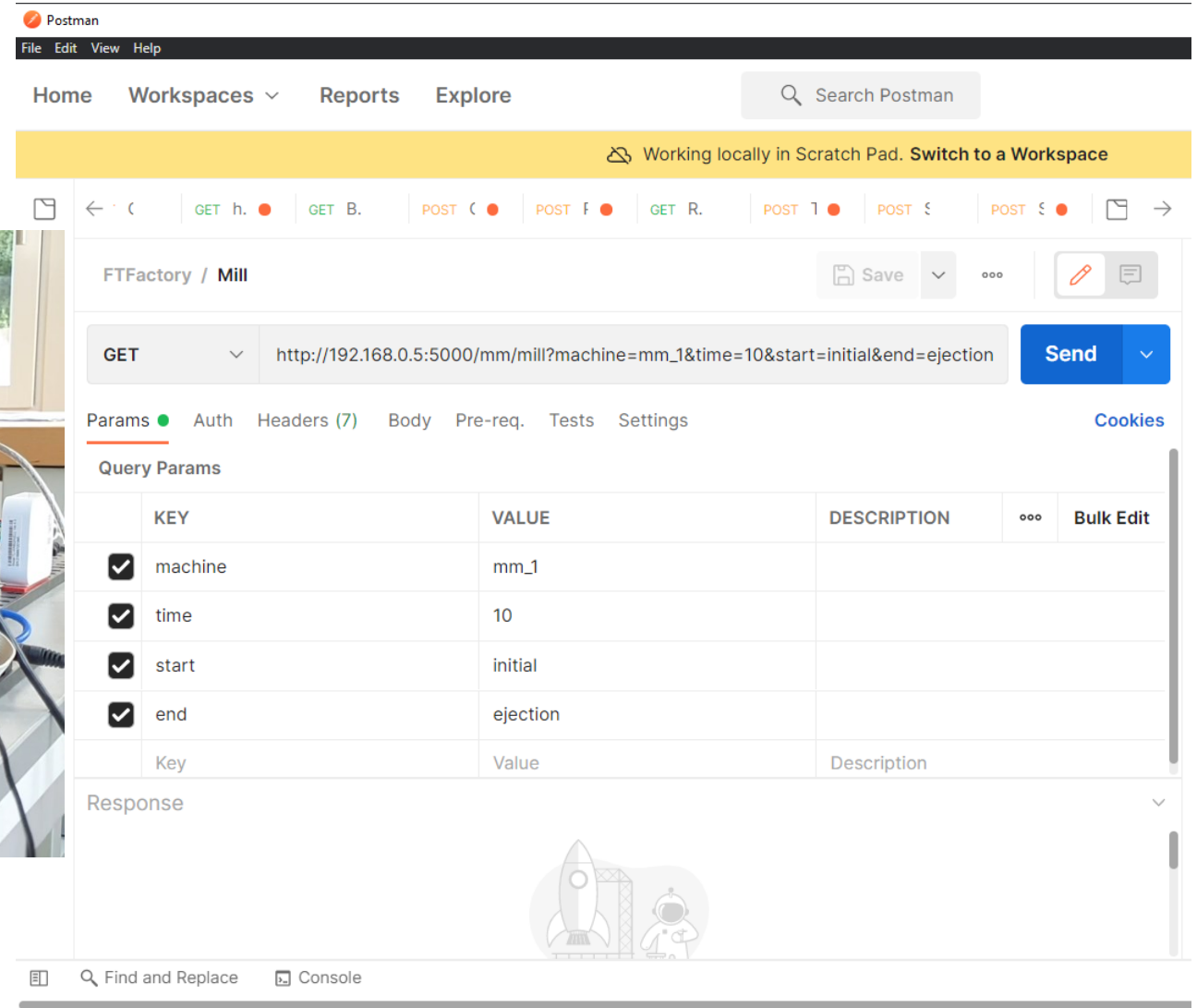
**GET** /mm/mill Starts milling.

If the workpiece moves to the milling machine, mills and then moves to the ejection position, pushes the workpiece onto the conveyor belt, starts it for 10 seconds and then returns to the initial position.  
 mm/mill?machine=mm\_1&time=10&start=initial&end=ejection [Example Link](#)

Parameters Try it out

| Name  | Description   |
|---|---|
| <b>machine</b> * required<br>string<br>(path) | mm_1<br><input type="text" value="mm_1"/>                             |
| time<br>integer<br>(path)                     | time<br><input type="text" value="5"/>                                |
| <b>start</b> * required<br>string<br>(path)   | Start Position<br><input type="text" value="start - Start Position"/> |
| <b>end</b> * required<br>string<br>(path)     | End Position<br><input type="text" value="end - End Position"/>       |

## Going Service-based: Example

Postman interface showing a GET request configuration for a service-based machine.

Working locally in Scratch Pad. [Switch to a Workspace](#)

FTFactory / Mill

GET `http://192.168.0.5:5000/mm/mill?machine=mm_1&time=10&start=initial&end=ejection` [Send](#)

Params **Auth** Headers (7) Body Pre-req. Tests Settings [Cookies](#)

Query Params

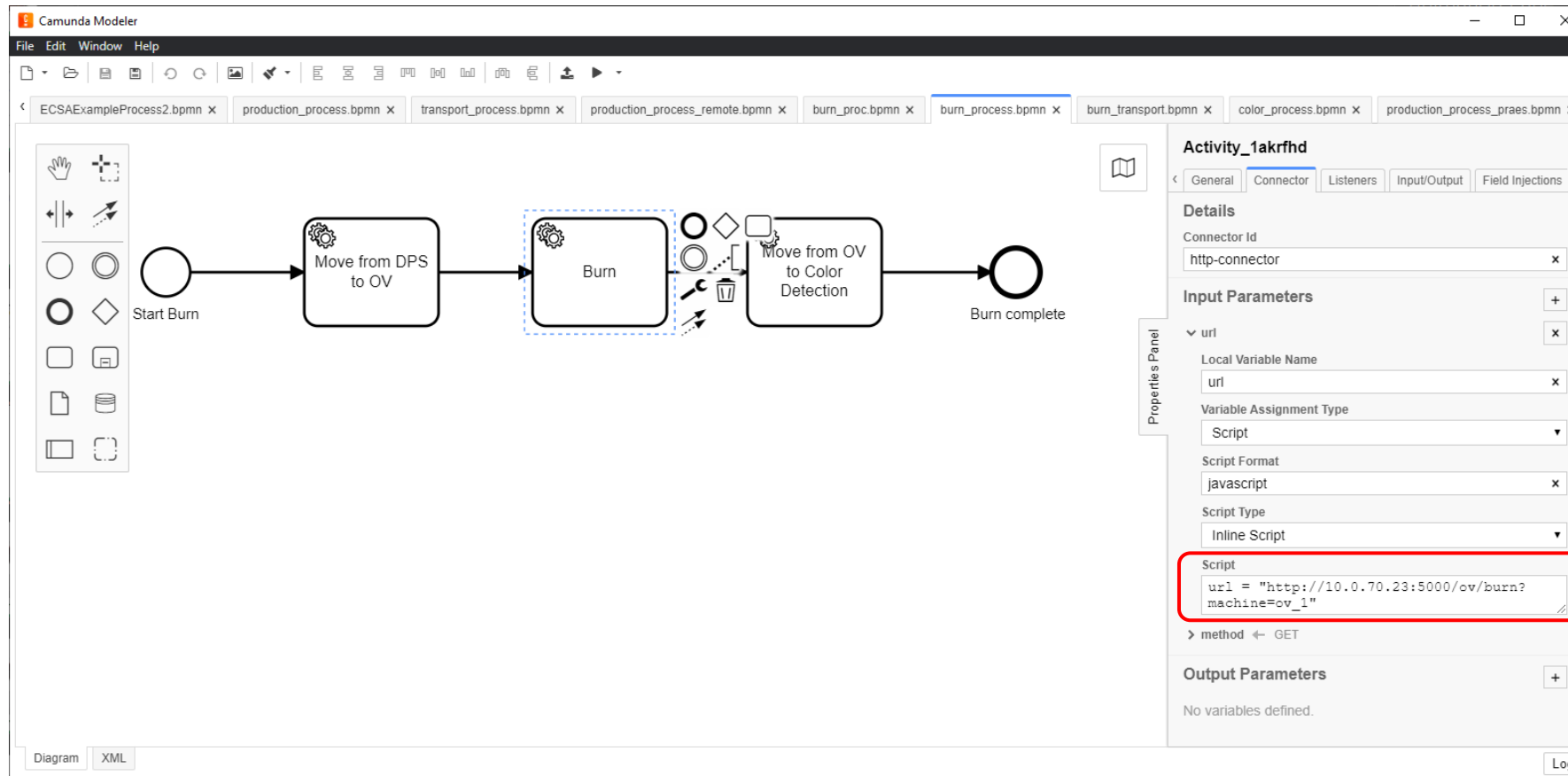
|                                     | KEY     | VALUE    | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|---------|----------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | machine | mm_1     |             |     |           |
| <input checked="" type="checkbox"/> | time    | 10       |             |     |           |
| <input checked="" type="checkbox"/> | start   | initial  |             |     |           |
| <input checked="" type="checkbox"/> | end     | ejection |             |     |           |
|                                     | Key     | Value    | Description |     |           |

Response

Find and Replace Console

## Going Business Processes

- Call web service via BPMN Service Tasks using http-connector



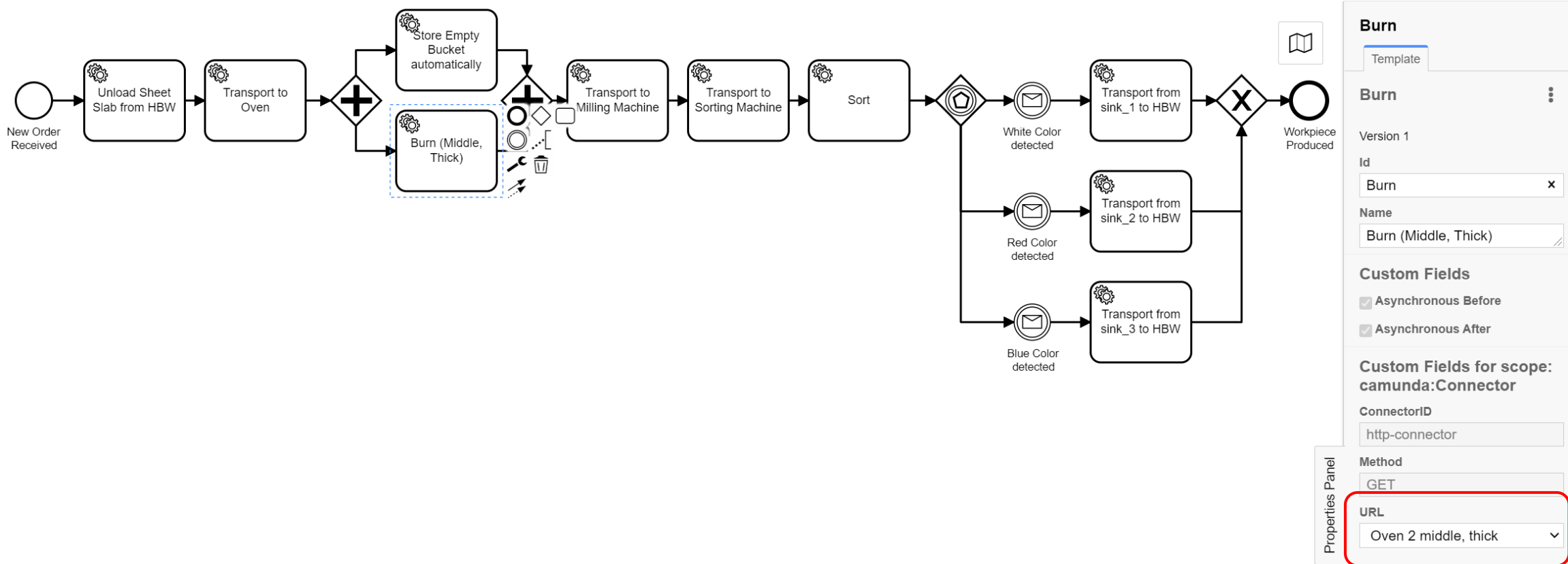
The screenshot shows the Camunda Modeler interface. The main diagram area displays a BPMN process flow: Start Burn (Start Event) → Move from DPS to OV (Service Task) → Burn (Service Task) → Move from OV to Color Detection (Service Task) → Burn complete (End Event). The 'Burn' task is highlighted with a blue dashed border. The Properties Panel on the right is open to the 'Connector' tab for 'Activity\_1akrfhd'. The 'Details' section shows 'Connector Id' as 'http-connector'. Under 'Input Parameters', the 'url' parameter is expanded, showing a 'Script' format with the following content:

```
url = "http://10.0.70.23:5000/ov/burn?
machine=ov_1"
```

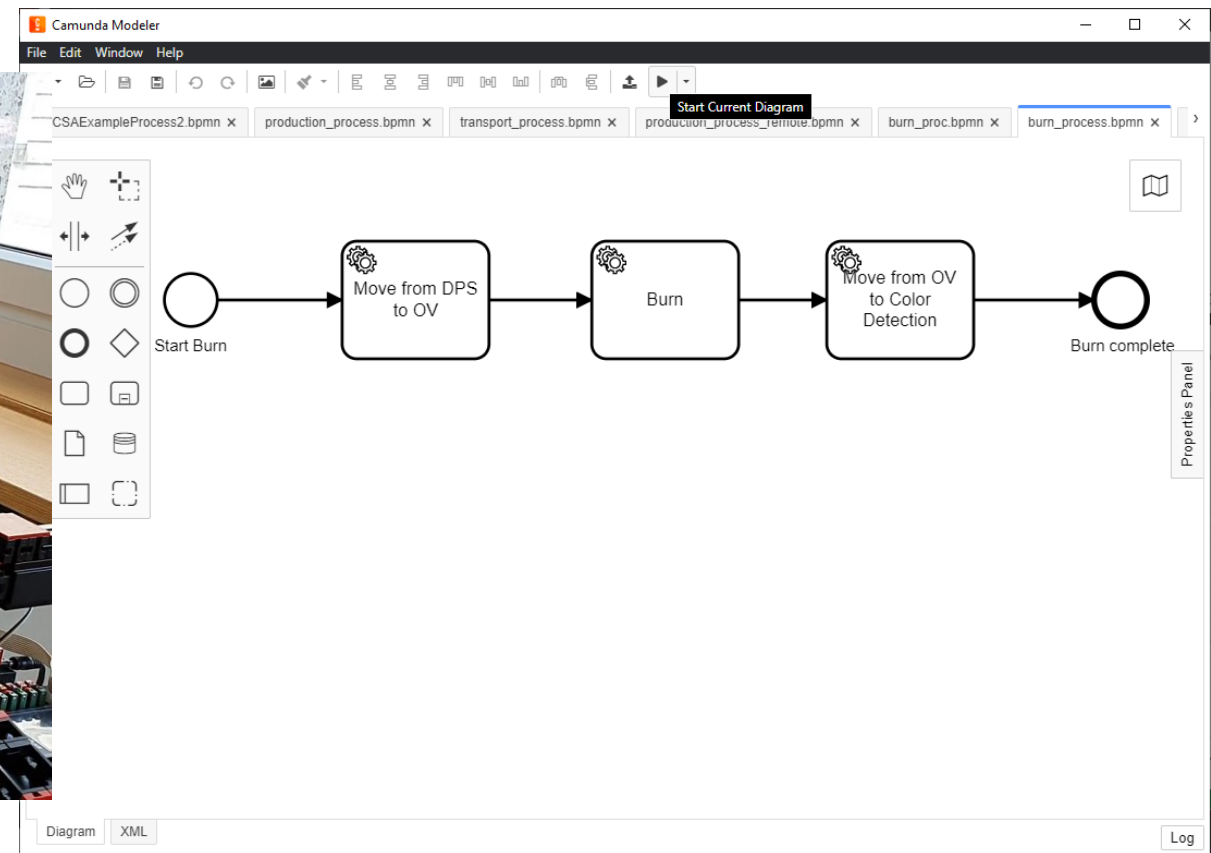
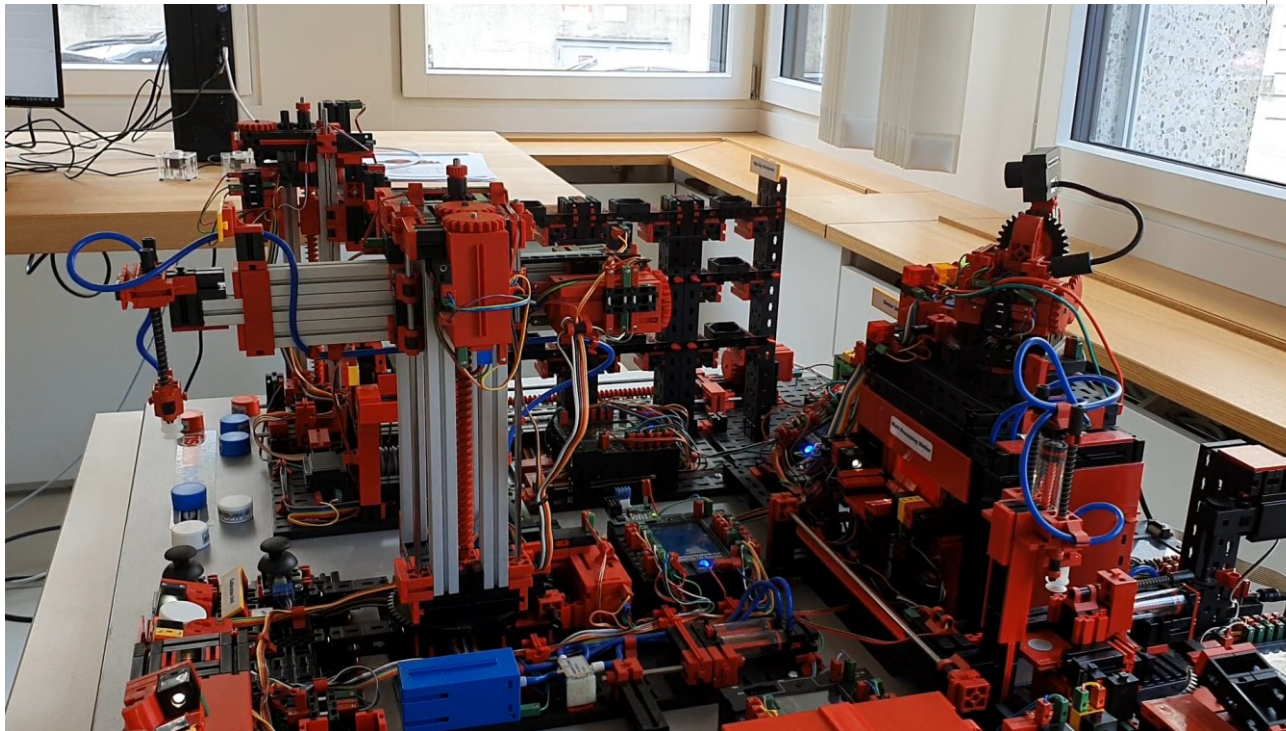
The 'method' is set to 'GET'. The 'Output Parameters' section is empty, showing 'No variables defined.'

## Going Business Processes

- Model processes with element templates

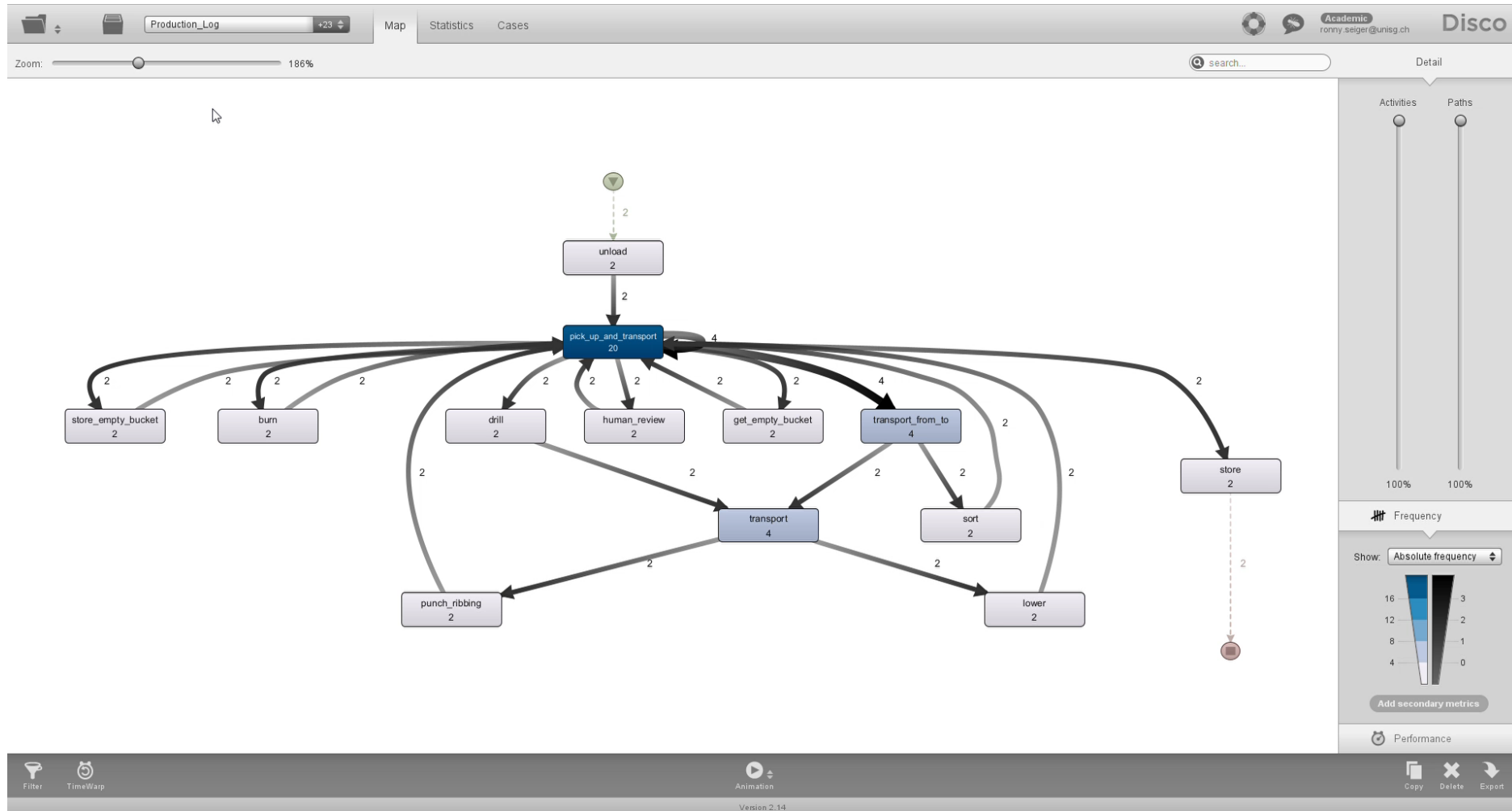


# Going Business Processes: Example

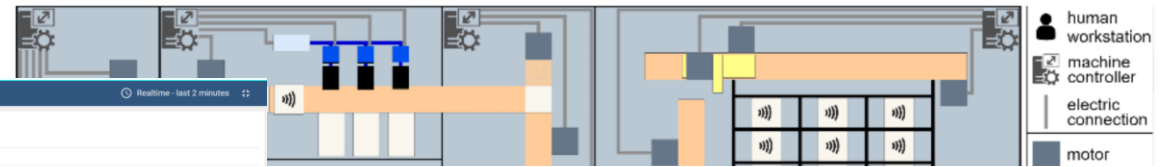




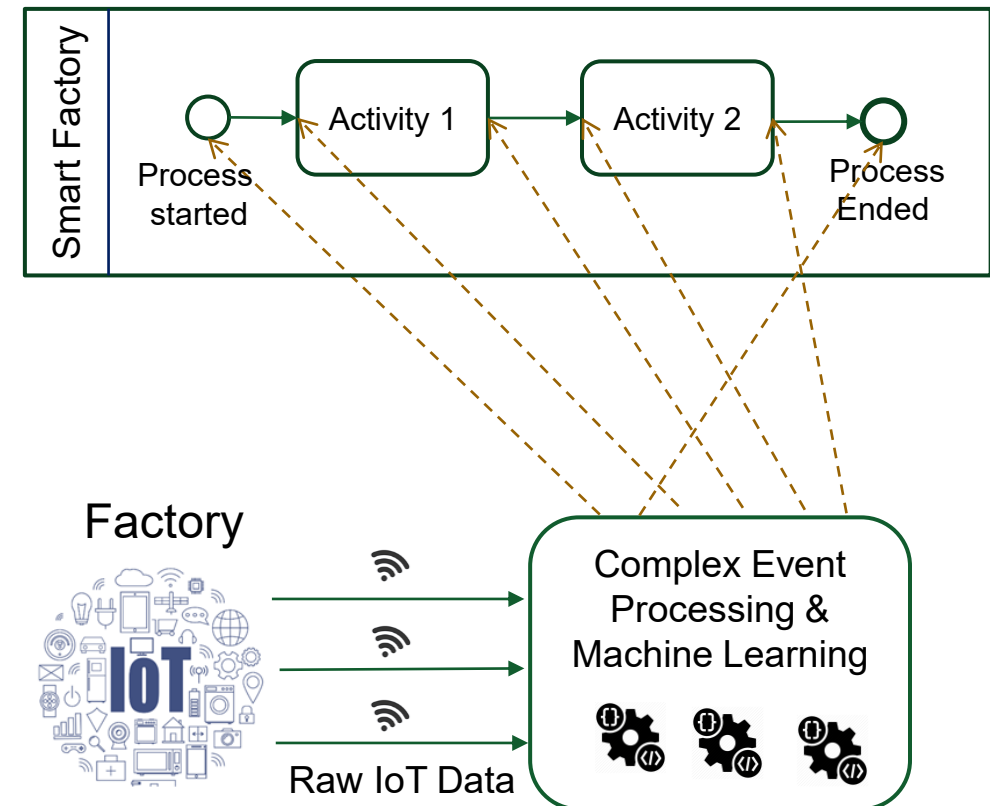
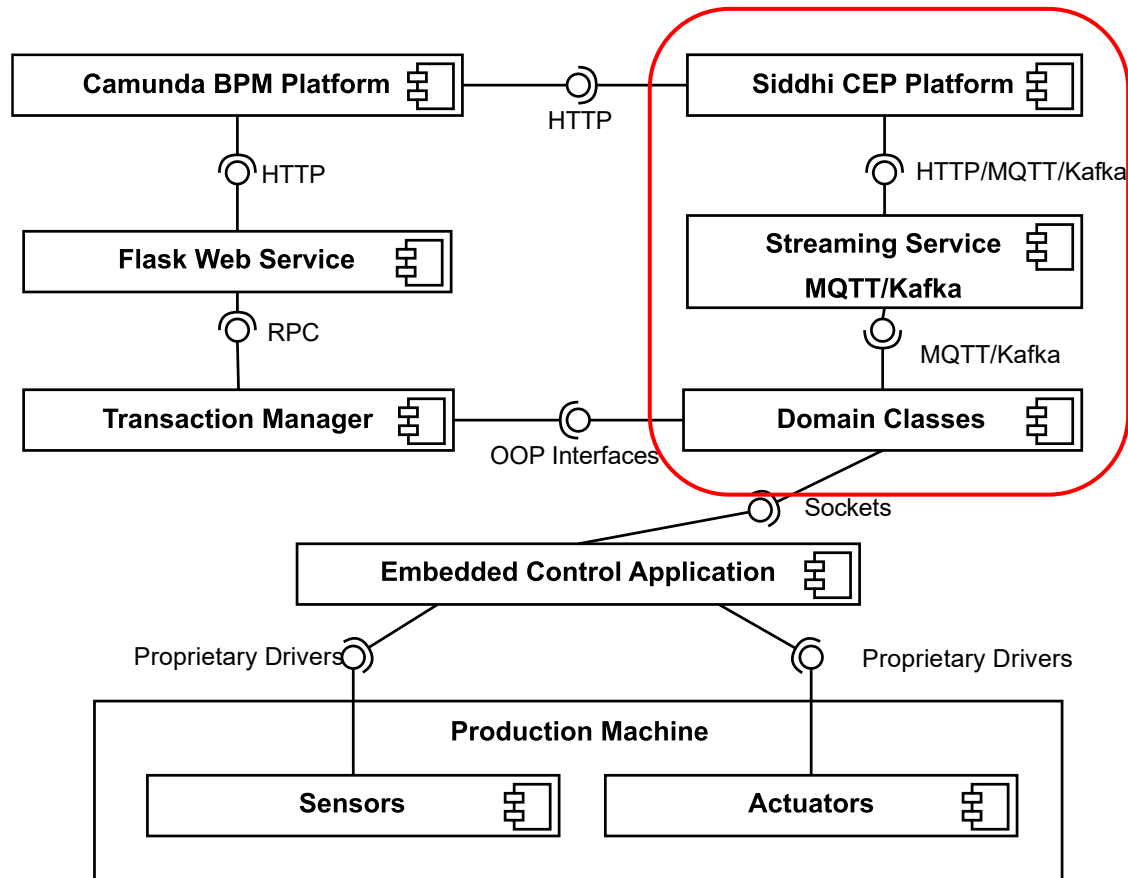
# Process Mining almost for Free



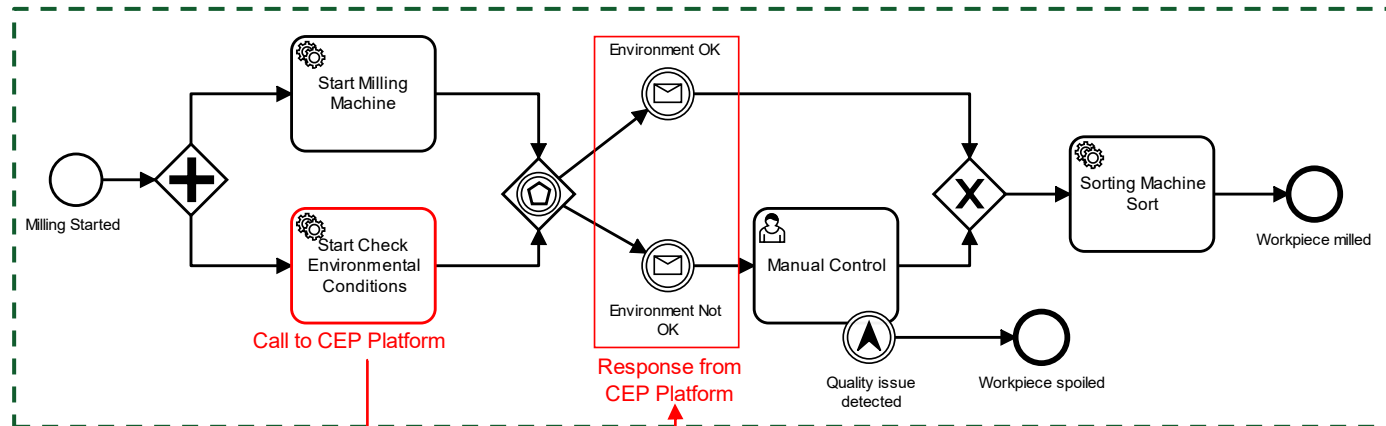
# What about Sensors?



## Going Event-driven

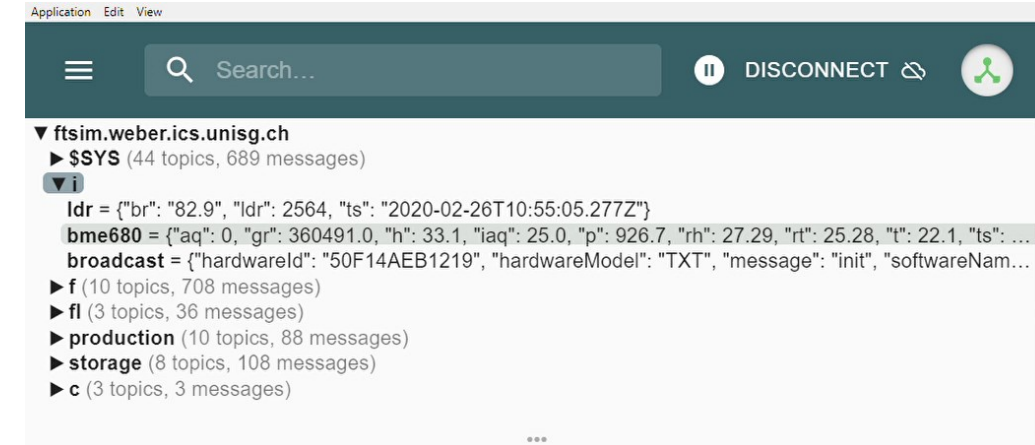


## Going Event-driven: Quality Check Example



Camunda BPMS

## MQTT Streaming



HTTP POST

HTTP POST

```
@info(name='EnvCheckOk')
from EnvAnalysisStream#window.timeBatch(10sec)
select "env_ok" as msgname, avg(t) as avgtemp,
      avg(br) as avglight
having (avgtemp >= 22 and avgtemp <= 25) and
      (avglight >= 80 and avglight <= 85)
insert into EnvCheckStream;
```

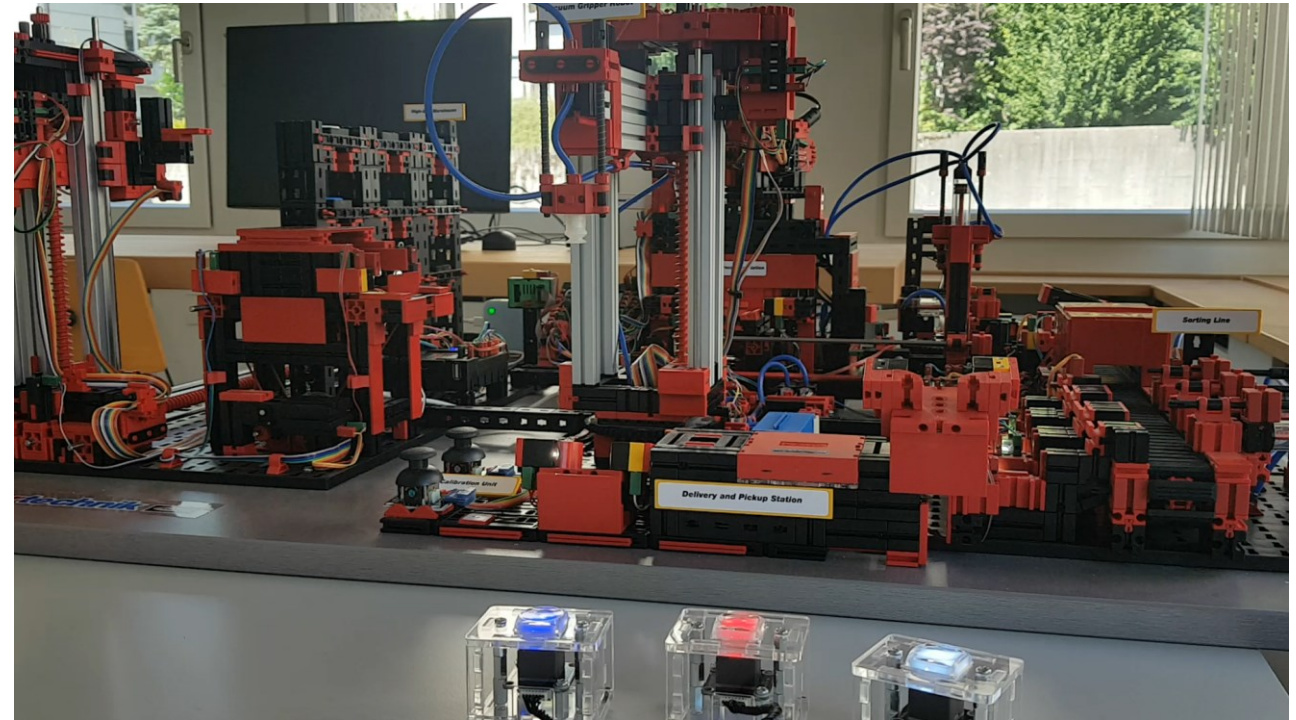
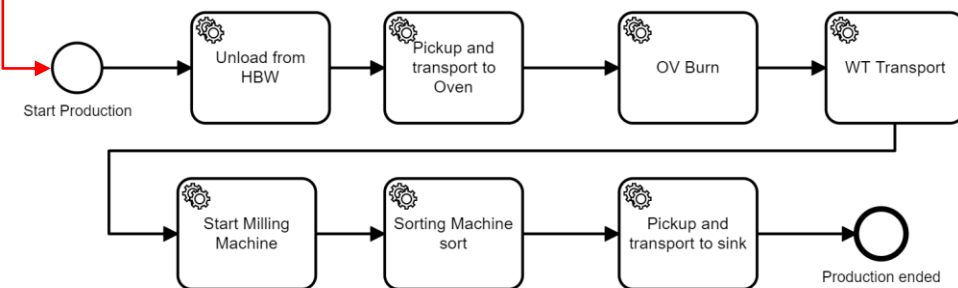
Siddhi CEP

## Going Event-driven: Starting a Process

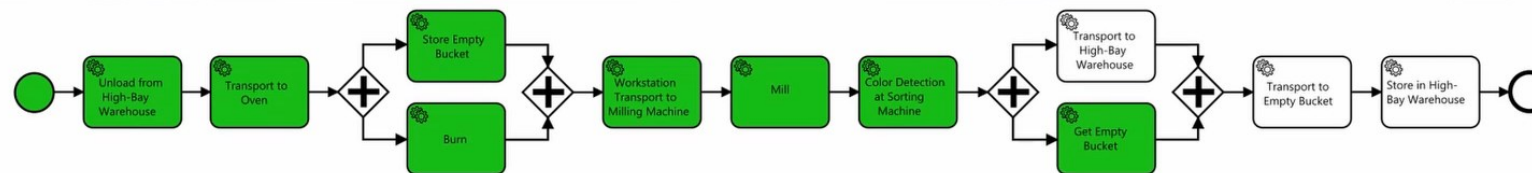
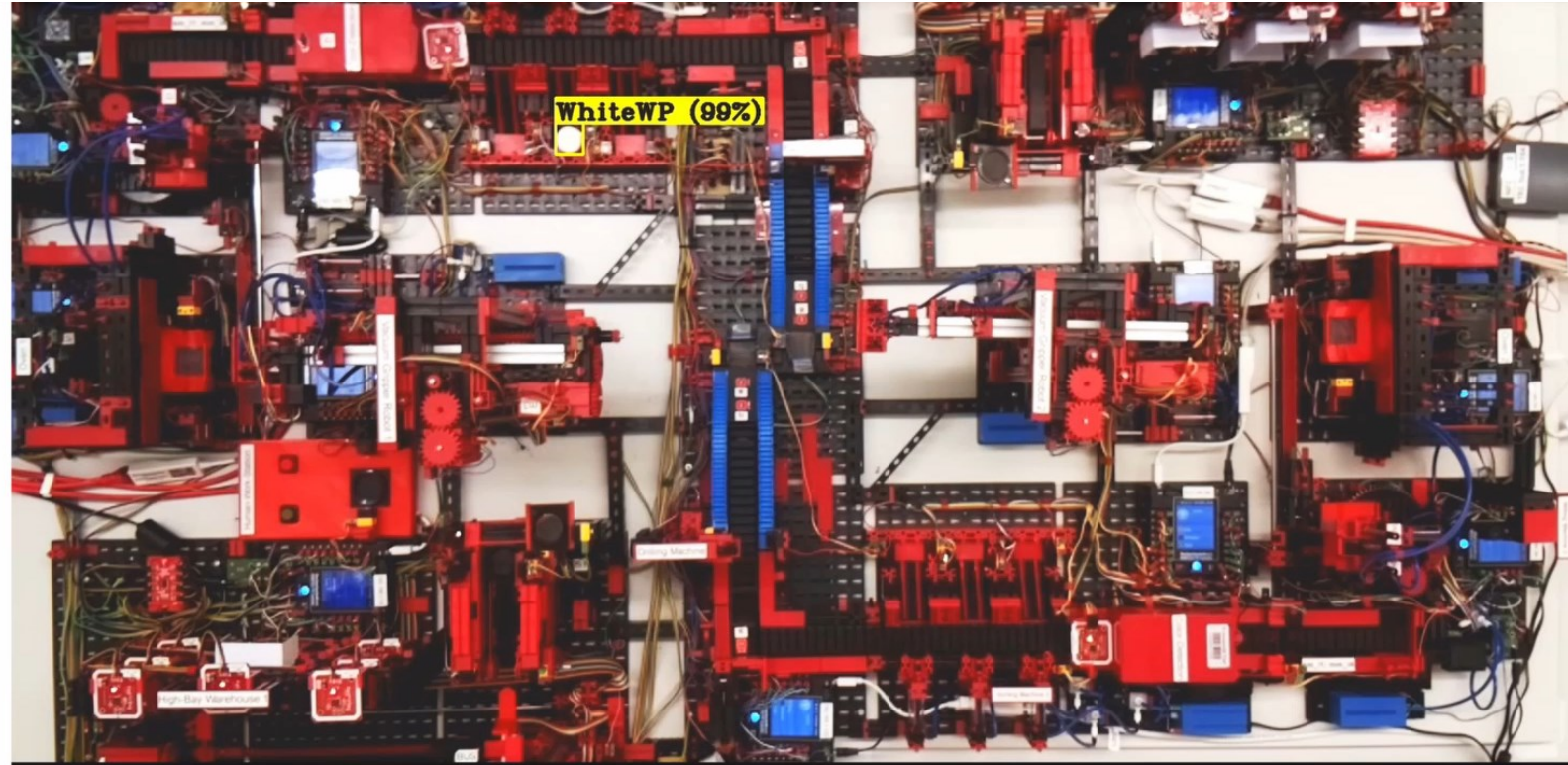
```
@source(type = 'mqtt', url =
"tcp://broker.hivemq.com", client.id = "unisg", topic =
"TF_Buttons/M3X", @map(type = 'json'))
define stream BlueButtonStream (type string, state
string, ts string);
```

```
@info(name='bluebutton')
from BlueButtonStream[state == 'pressed']
select str:lower(type) as color
insert into ProcessStartStream;
```

HTTP POST

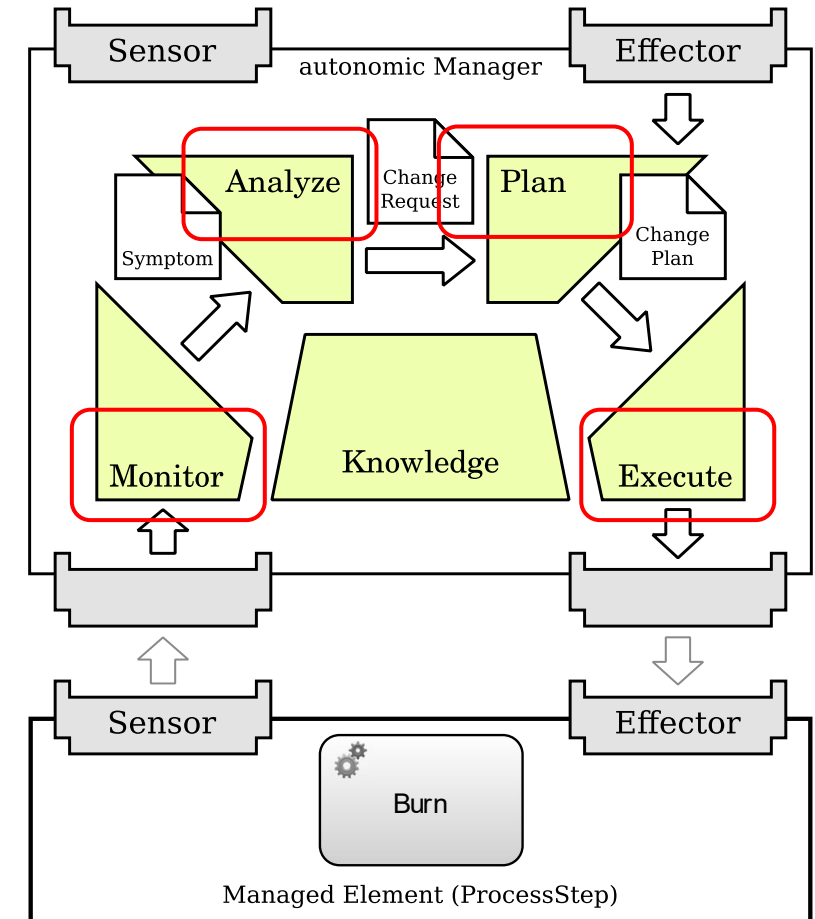


# AI is Everywhere!

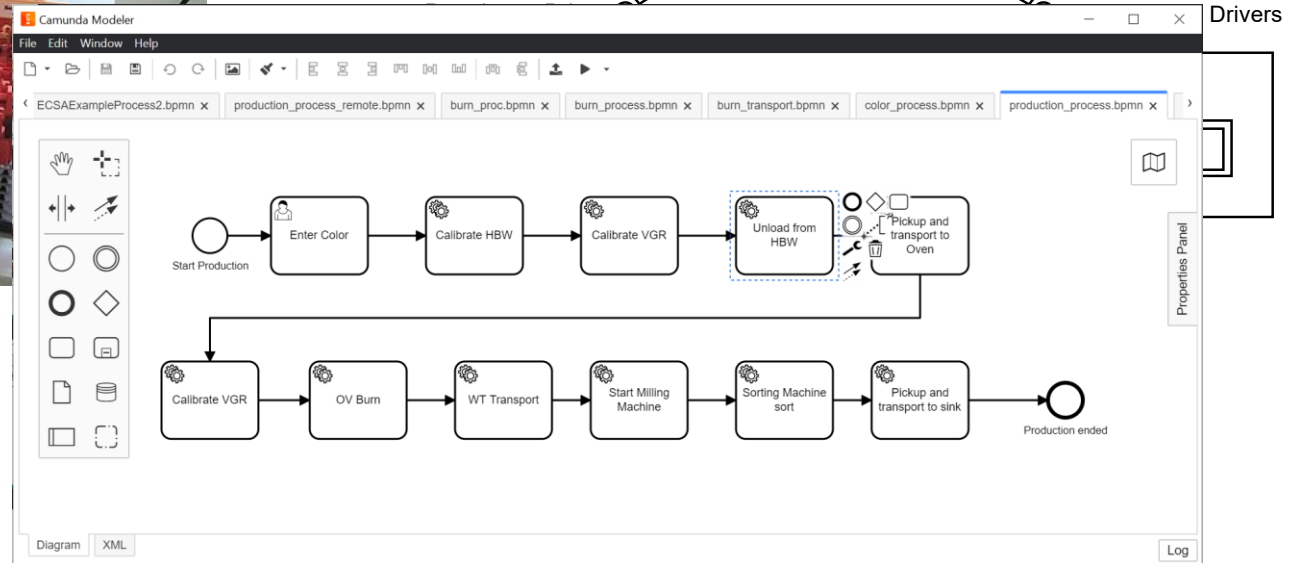
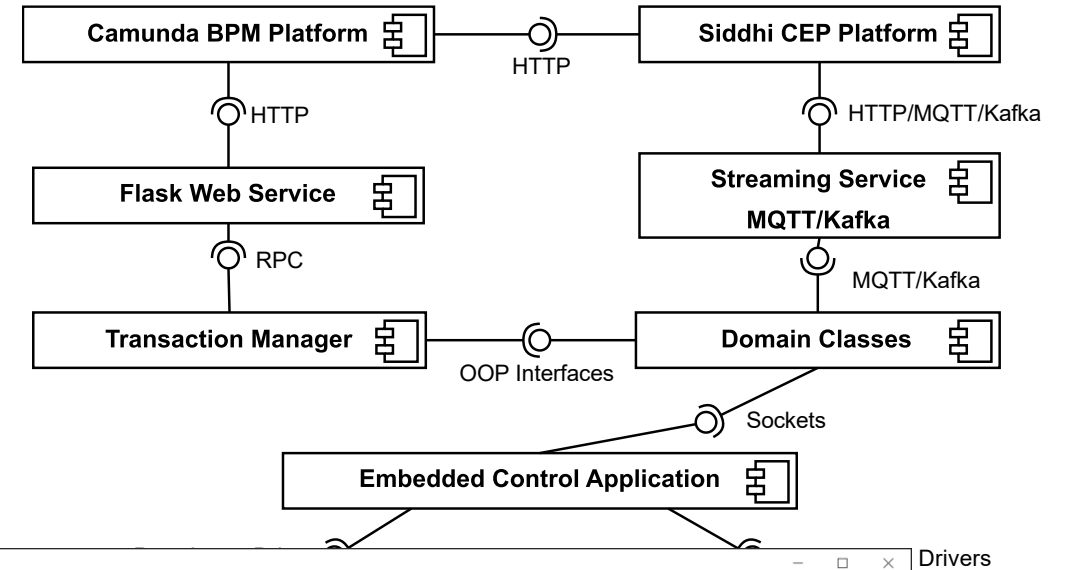
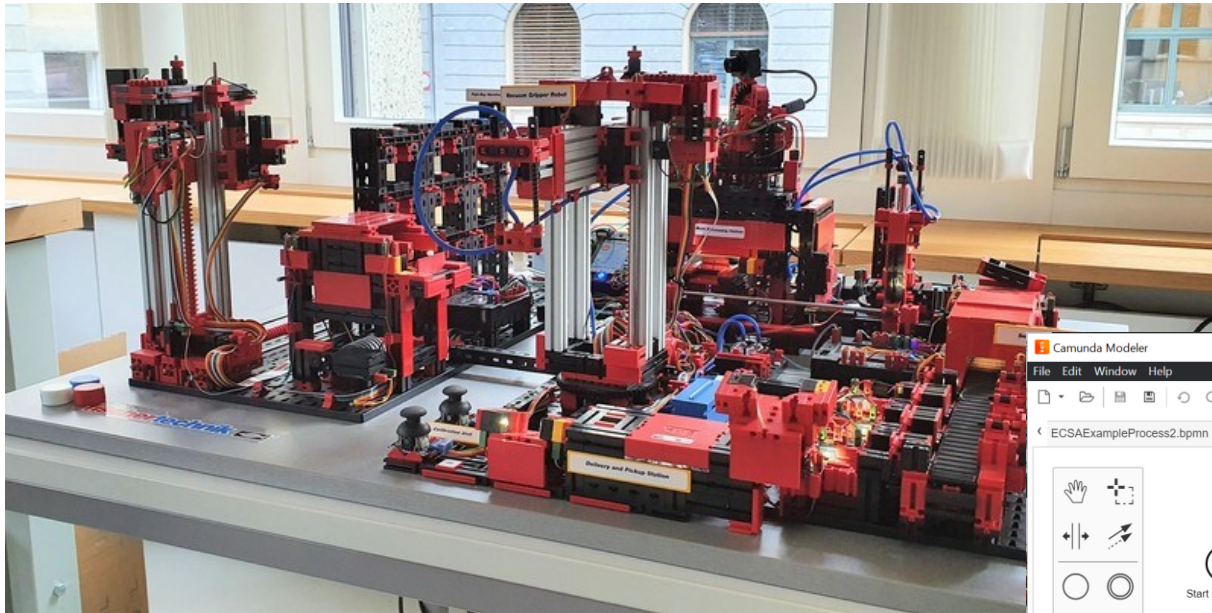


## Now the Research

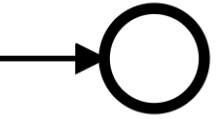
- **Error Handling & Autonomous Behavior**
- Monitor: Sensor Streaming
- Analyse: Complex Event Processing & Machine Learning
- Plan: Case-based Reasoning (Learn from the Past) & Automated (AI) Planning
- Execute: Adapt the Process
- Repeat



# The Whole Picture







Presentation  
finished

## Controlling a Smart Factory with Processes in Camunda

### Get in Touch

Ronny Seiger, Barbara Weber  
University of St.Gallen, Switzerland  
[ronny.seiger@unisg.ch](mailto:ronny.seiger@unisg.ch)

Lukas Malburg, Ralph Bergmann  
University of Trier, Germany  
German Research Center for AI (DFKI)  
[malburgl@uni-trier.de](mailto:malburgl@uni-trier.de)





# Questions?