Demo Implementation of Industrial IoT Factory Floor based on Arduino and Raspberry Pi

Project Acronym	IoT4Industry
Document-Id	D.6
File name	
Version	FINAL document
Date	Start: 01 April 2015 End: 31 September 2015
Author(s)	Robert Mulrenin Felix Strohmeier Oliver Jung
QA Process	Georg Güntner Dr. Violeta Damjanovic-Behrendt

Table of Content

IoT4Industry in a Nutshell

IoT4Industry is an exploratory project funded by the Austrian Research Promotion Agency, for the period October 2014 – September 2015. It stands for "Secure, Privacy-preserving Agents for the Industrial Internet". The core research areas of IoT4Industry relate to security, privacy and IPR/data protection requirements, translated into game-theoretic agent behavior, which is simulated in a demo factory floor environment, supporting a supply chain negotiation. A demo factory floor is implemented within our IoT-lab, which is located at Salzburg Research.

While it is still not clear which protocols and technologies will become mainstream for the Industrial Internet, it has become clear that security and privacy will feature strongly in any risk assessment concerning the adoption of practices using the Industrial Internet. Thus, in IoT4Industry, we focus on the use of policy-enacting, multi-agent systems that securely manage machines and manufacturing cells, and we build a feasibility demonstrator based on open source tools and firmware. In addition, IoT4Industry helps us to prepare for internationally recognized contributions to science and technology in the field of Industrial Internet, notably in Horizon 2020.

Note: This report includes the description of work that was planned to be presented in D.4 “Setup of a Demo Factory Floor based on Arduino and Raspberry Pi”. We decided to describe the work done in tasks T.4 and T.6 in one joint report that we call D.6 “Demo Implementation of Industrial IoT Factory Floor based on Arduino and Raspberry Pi”.

IoT4Industry Task 6 Description

(from the *IoT4Industry* project proposal)

D6: Demo Implementation of Industrial IoT Factory Floor based on Arduino and Raspberry Pi

Goals:

1. Web-observable manufacturing scale models with sensors, managed by IoT Agents; 2. Technical Report on software, firmware and hardware setup (open access, open source)

3. Data repository online, for external replication of results

Description of the content

We “polish” the system so that it is available and usable on-line, as a feasibility demonstrator, accessible via an API and so that data sets relating to publications are freely accessible.

Method

Improvements in the code base, its documentation and usability

Write-up of a “How-to” on project wiki (e.g. github)

Milestones, results and deliverables

09/2015: Online demo, API and data repository together with manuals, are available

1. Introduction

Due to a great technical dependency between tasks T4 (demo setup) and T6 (demo implementation), we decided to merge our experience gained during implementation of these two tasks within one report, D.6 “*Demo Implementation of Industrial IoT Factory Floor based on Arduino and Raspberry Pi*”. In order to reduce the workload and obtain sensor data earlier in the project, we additionally decided to move from Lego-technic and Fischer-Technik showcase models, as it was planned in the project proposal, towards implementing a more concrete environment for additive manufacturing (3D printing). Our experimenting IoT-Lab, at Salzburg Research, was already equipped by many sensors, and we wanted to make a use of them for the purpose of *IoT4Industry* experimentations. Hence, our industrial scenario setup implements a 3D printer farm, which has been extended by external sensors and actuators to improve not only automation, but also security aspects in *IoT4Industry*. This report describes the entire 3Dp printing setup, including relevant components, protocols and technologies used.

2. Demo Scenario

In order to analyse and simulate the industrial challenges in our laboratory setup and build upon existing business models, we decided to implement a 3D printing farm, and to enhance it with novel ways of data acquisition. This chapter describes the demo scenario and data acquisition approaches.

2.1 Description of the Demo Scenario

Our experimental laboratory setup includes one 3D printer and a set of sensors, which are described in detail in the data acquisition section. Furthermore, 3D printers and their corresponding sensor sets are simulated in order to approximate the printing farm scenario. An overview of the scenario is depicted in Figure 1.

The central controller of the cyber-physical system (CPS) is at the heart of the control flow. Furthermore, sensors, agents and actuators are part of the CPS. First, the sensors submit their sensor data to the controller where the data is filtered and processed. Afterwards, the controller distributes relevant data overviews to the different agents. The agents evaluate the data, report back to the controller and give instructions to the actuators. Actuators perform the corresponding actions and report back to the agents.

A repository from Figure 1 serves as a virtual image of the entire CPS.

The users of the CPS are divided into three groups: the customer, the service technician and the manufacturer. The **customer** has limited interaction (based on his trust level) with the controller and can retrieve his own order overview, as shown in Figure 2.

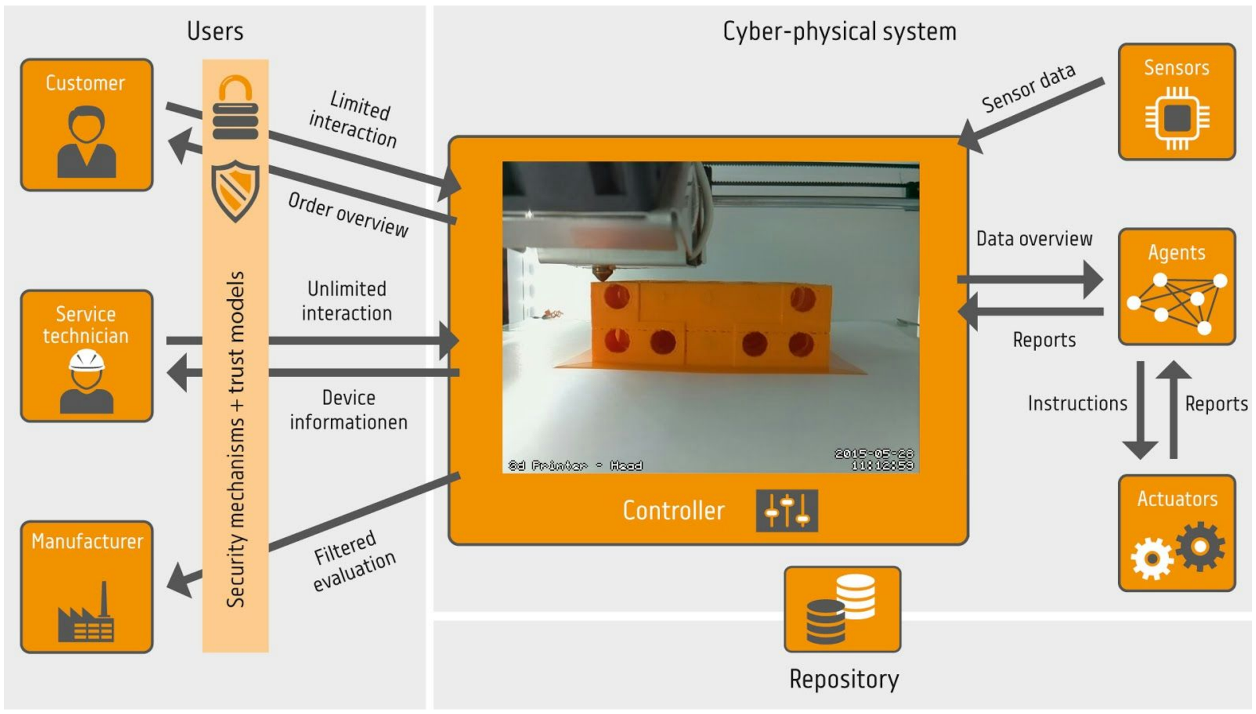
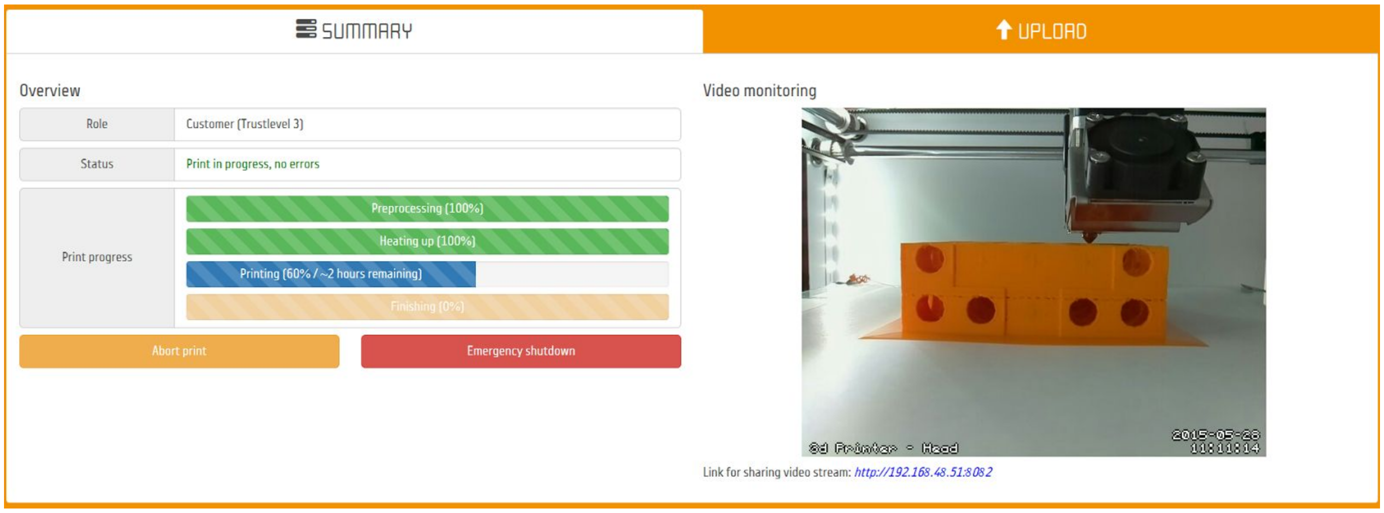


Figure 1. Scenario overview



Overview

Role	Customer (Trustlevel 3)
Status	Print in progress, no errors
Print progress	Preprocessing (100%)
	Heating up (100%)
	Printing (60% / ~2 hours remaining)
	Finishing (0%)

Video monitoring

Link for sharing video stream: http://192.168.48.51:8082

Figure 2. Customer interface

The **service technician** has unlimited interaction with the controller and can retrieve relevant device information for maintenance purpose, triggered by the controller itself or by a customer. Furthermore, he can manage trust levels (see Figure 3).

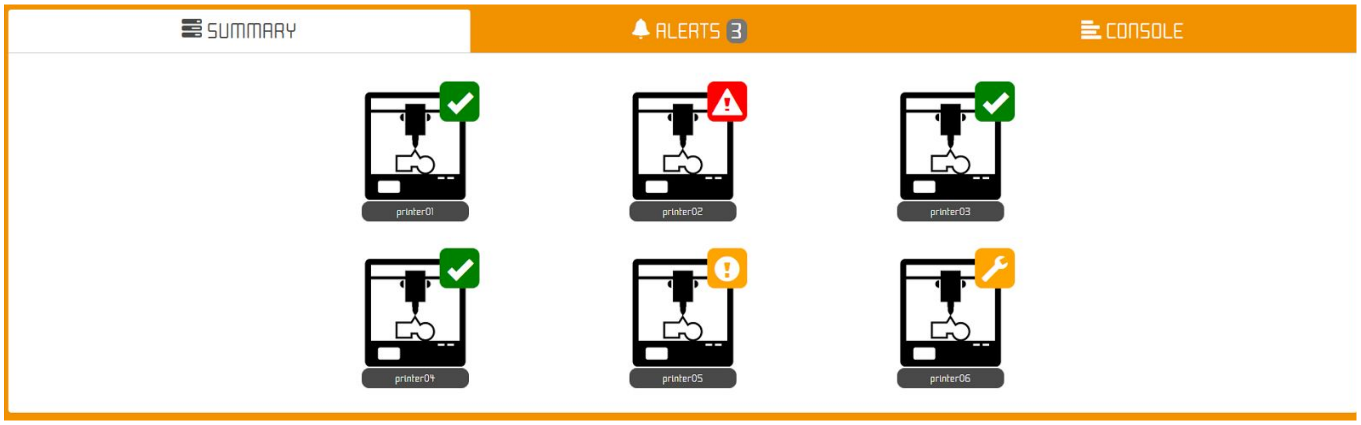
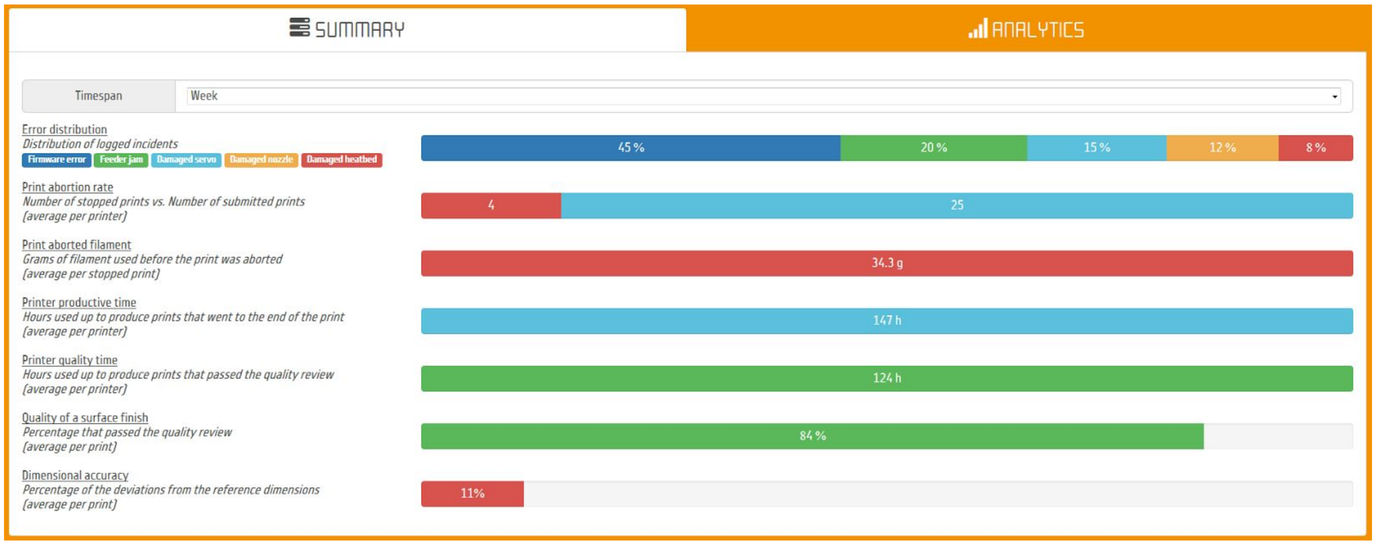


Figure 3. Service technician interface

Finally, the **manufacturer** may receive a filtered and anonymized evaluation of device usage over different time periods and analytics in terms of derived data and predictions (see Figure 4).



Timespan: Week

Error distribution
Distribution of logged incidents
Firmware error, Feeder jam, Damaged servo, Damaged nozzle, Damaged heatbed
45 % | 20 % | 15 % | 12 % | 8 %

Print abortion rate
Number of stopped prints vs. Number of submitted prints (average per printer)
4 | 25

Print aborted filament
Grams of filament used before the print was aborted (average per stopped print)
34.3 g

Printer productive time
Hours used up to produce prints that went to the end of the print (average per printer)
147 h

Printer quality time
Hours used up to produce prints that passed the quality review (average per printer)
124 h

Quality of a surface finish
Percentage that passed the quality review (average per print)
84 %

Dimensional accuracy
Percentage of the deviations from the reference dimensions (average per print)
11%

Figure 4. Manufacturer interface

All interaction is controlled by means of security mechanisms, such as encryption and authentication, and trust models. The relevant approaches and processes will be described later on.

2.2 Data Acquisition

There are various 3D printing features that can be retrieved directly from the printer controllers, external sensors, or can be derived from aggregated data. The most important features and their sources are listed in the table below.

3D printing feature	Data source
Printer status	Central controller and external sensor (remote power switch)
Print progress	Printer controller, Filament Sensor
Nozzle and heatbed position (x,y,z)	Printer controller
Nozzle and heatbed temperature	Printer controller
Printer material settings (e.g. material type, material flow, fan speed)	Printer controller
Print specific settings (e.g. material fill, print speed, support structure type)	GCode file

Filament consumption	Estimated from GCode file; Real data from external sensor (light barrier measurement)
Print duration	Estimated from GCode file; Real data from central controller
Print quality	Estimated from external sensor (camera); Customer Feedback; Filament Sensor
Nozzle acceleration	External sensor (accelerometer)
Power consumption	External sensor (split-core current transformer)
Air pollution	External sensor (air quality sensor; aggregated value)
Room temperature	External sensor (temperature sensor)

2.2.1 3D Printer "Ultimaker 2"

The core device in our 3D printing farm is the "Ultimaker 2", a 3D printer with "Fused Filament Fabrication" technology[1], shown in Figure 5. It can create/print 3D-models from PLA and ABS plastics, with a maximum size of 230 x 225 x 205mm (width x depth x height). The complete Ultimaker 2 design is available as open source. Therefore files for spare parts can be downloaded and the printer can be modified/extended where appropriate.

[1] https://ultimaker.com/en/products/ultimaker-2-family

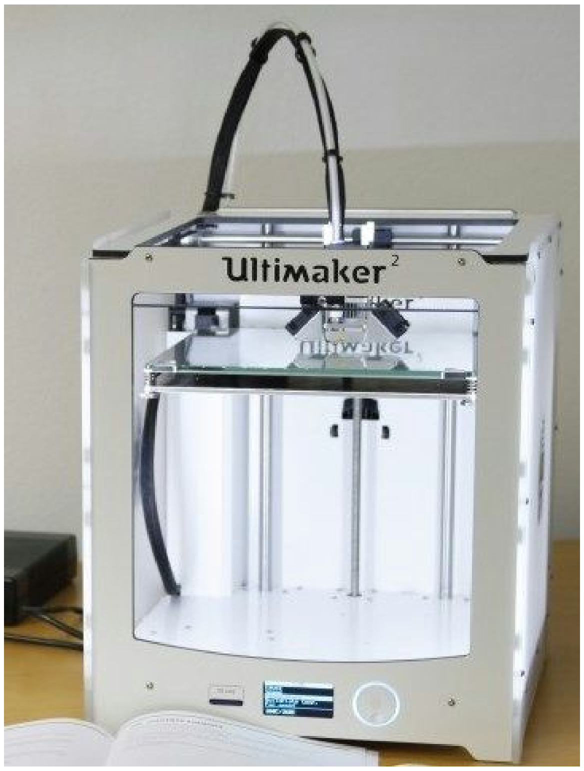


Figure 5. Ultimaker 2

2.2.2 Printer Controller

Ultimaker 2 was originally designed for stand-alone SD card printing. To enable remote printing control via Internet protocols, we added an Internet-enabled low-performance device to serve as an external controller. Different ready-made products such as “Doodle3D”[2], “OctoPrint”[3] or “AstroPrint”[4] are available, however all of them rely on a reliable USB-Connection to the 3D printer, which is not supported by Ultimaker 2. We therefore implemented our own controller for uploading and slicing 3D models, queuing print jobs, turning the printer on and off (via a remote power control unit), and also receive status information from the printer.

The external printer controller is located on a Raspberry Pi, which is connected via USB to the internal control unit of the printer (an Arduino-based board inside the Ultimaker 2). To get a reliable serial communication via USB, we built on a specialised Ultimaker 2 firmware and printing utility called “Ultiprint”[5]. Via the USB-Interface the printer can now receive commands from the external controller and upload the 3D model files (in G-Code) to the SD card of the

[2] http://www.doodle3d.com/
[3] http://octoprint.org/
[4] https://www.astroprint.com/
[5] https://github.com/ErwinRieger/Ultimaker2Marlin-USBPrint/wiki/UltiPrint.py-Utility

printer. During the running print-job, it is used to receive information about the following status information from the printer: x/y/z-position, extrusion, as well as temperatures from the heat-bed and the nozzle. Finally, the major task of the external printer controller is to provide Internet-connectivity for making the printer-related web-services accessible to outside software agents and human users.
The external printer controller is shown in Figure 6. It is connected to the printer (via USB) (1), the remote control for the power switch unit (2) and a prototyping breadboard with status LEDs and a button for (limited) human interaction (3). The interaction showing complete printing functionality is also accessible through the REST-based web-services.

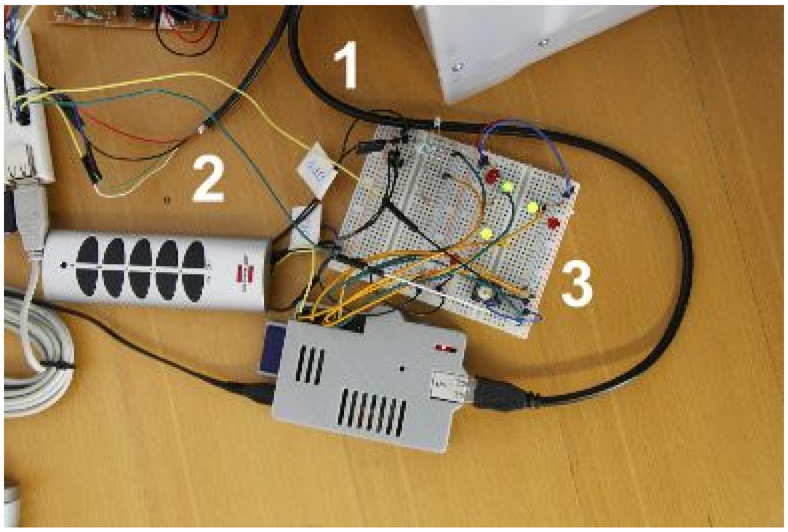


Figure 6. External printer controller on a Raspberry Pi

2.2.3 Accelerometer

The accelerometer is a “LIS344AL”, three-axis accelerometer from STMicroelectronics. It is mounted on a TinkerKit module, which outputs volt-levels between 0V and 5V. The output of the accelerometer is connected to an analogue filter and an analog-digital converter and fed to a Raspberry Pi that connects the sensor to the Internet. To enable a precise measurement of all accelerations, we use an active 2nd-order Chebyshev filter with a ripple of 3dB at a cut-off frequency of 500Hz. As shown in the Figure 7, the accelerometer module is mounted with a custom-made chassis on the printer-head of the Ultimaker.

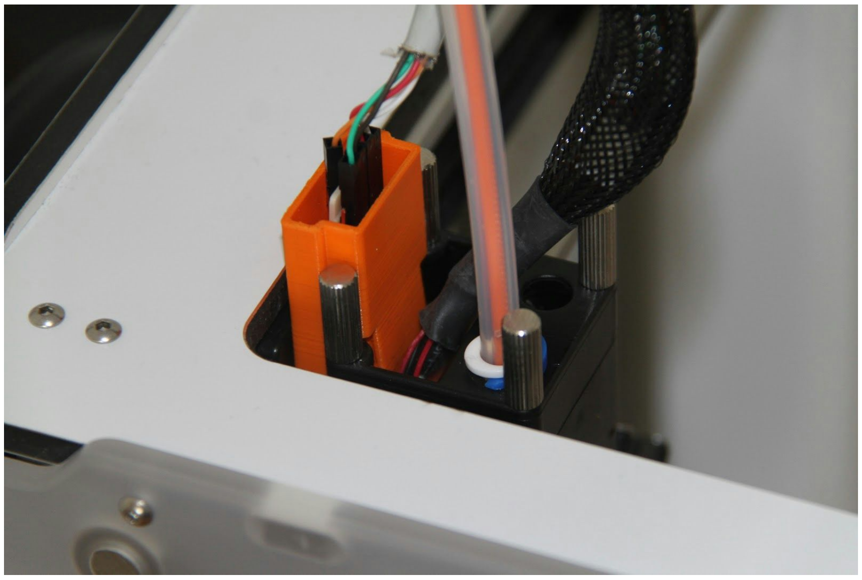


Figure 7. Accelerometer

2.2.4 Split-core current transformer

The split-core transformer is used to roughly measure the power consumption of the 3D printer. It is connected to a sensor box provided by a 3rd-party company “LineMetrics”, as depicted in Figure 8. The LineMetrics box directly sends its measurement data to their own cloud service, which can be accessed remotely via a web-interface.

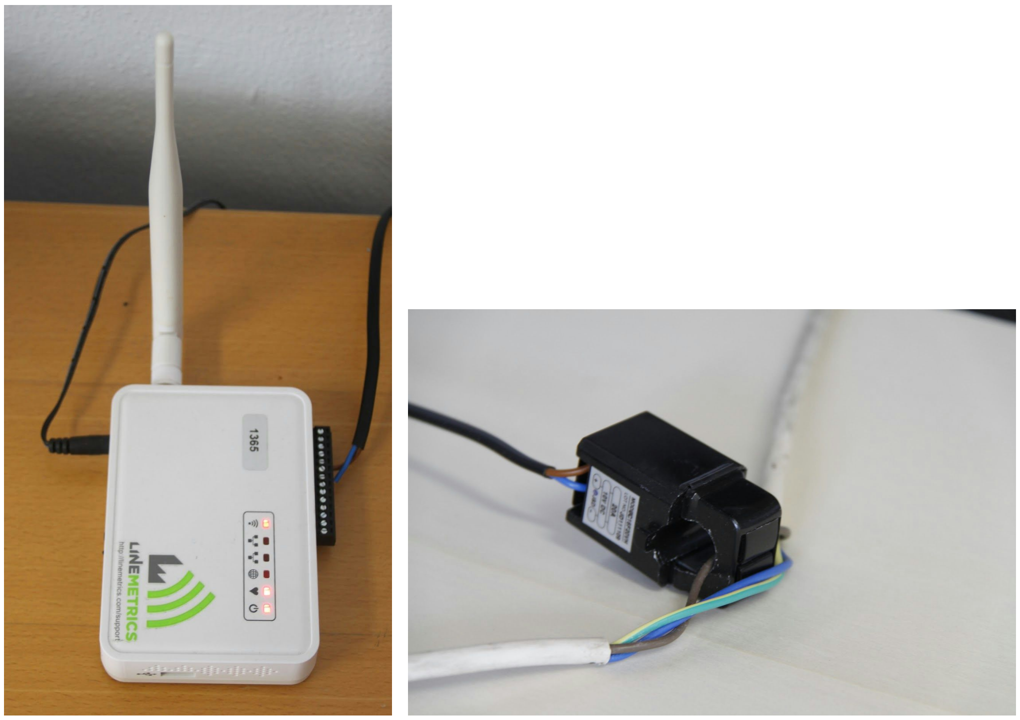


Figure 8. Power consumption measurement unit

2.2.5 Air Quality Sensor

The air quality sensor from Velux[6] is connected via a USB-Hub to the same Raspberry Pi as the accelerometer. It measures a combined air quality level on the basis of volatile organic compounds (VOCs) to provide a rough indicator on the current air quality without exact details on the measured gases. As output, it provides a number on a scale between 450 (good) and 2500 (bad). A value between 1000 and 2000 indicates medium air quality. A LED indicator on the back of the sensor shows the quality in the colors red, yellow and green. Figure 9 shows a picture of the connected sensor (color indicator shows yellow = medium air quality).

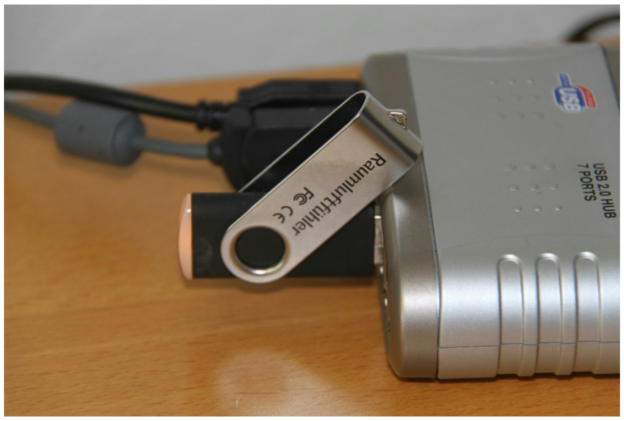


Figure 9. Air quality sensor

2.2.5 Filament Sensor

During the 3D printing process, the correct forwarding of the filament plays an important role to achieve good quality prints. Similar to the ink of a 2D inkjet printer, the filament is now the basis of any successful print. It is delivered as thin[7] plastic wire on spools that can be mounted to the back of the printer. Via the so-called “extruder”, a forwarding device driven by a stepper motor and a bowden tube, the material is feed to the heated nozzle. The nozzle is heated up to the melting point of the plastic as in a “hot-melt gun”. Printing quality is decreased or whole prints can be destroyed upon issues with the filament forwarding. Stucking material may have multiple reasons, such as low nozzle temperature, dirty nozzle, problem in the bowden tube that connects the feeder to the nozzle, stucking material spool, empty material, etc.

[6] http://www.velux.de/produkte/lueftungsloesungen-belueftung/raumluftfuehler
[7] For the standard nozzle in the Ultimaker 2 printer the material wire has a diameter of 2.85mm

We therefore designed a separate sensor that can be mounted to the extruder of the 3D printer to measure the input rate of the filament. As depicted in Figure 10, the components of the filament sensor are themselves printed using the 3D printer plus a rubber-band. This enables us to easily share the design and production of the components with other users of the same 3D printer and simplifies the reproduction of spare parts.

For sensing the filament movement, we use the principle of a rotating disc optical chopper. For the electronics inside the case, we use two simple photo eye sensors[8] to implement an incremental rotary encoder. This enables us to measure filament movements in both forward and backwards directions.

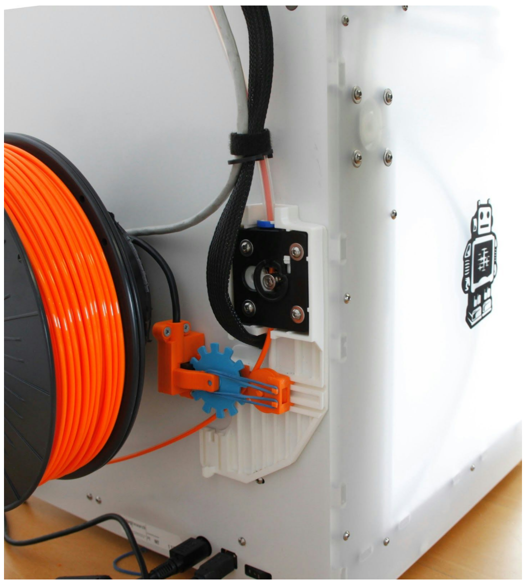


Figure 10. Filament Forward Sensor

2.3 Data Exploitation

The collected sensor data is assembled in different sensor nodes, and transferred to the central repository for further data processing, allowing for faster and smarter decision making, and improving complex industrial tasks.

[8] “Transmissive Optical Sensors with Phototransistor Output”, e.g. TCST 2103

We distinguish between two major approaches in data analytics, both of them can be exploited by software agents, as described in D.5 and also by other applications, e.g. by visualisation tools:

- **Real-time data analytics:** Data streams generated from the sensors are streamed in real-time using messaging protocols, such as MQTT to a central message broker. Using the publish/subscribe paradigm, analytics applications can subscribe to particular data streams for real-time analytics, decision making or visualisation.
- **Batch data analytics:** In addition to the real-time analytics components, sensor data is also streamed to a central data repository, implemented as a scalable cluster application, which can be accessed by applications or to perform historical data analytics on demand or in predefined intervals.

The concrete implementation technologies used are described in Section 3.

3. Technology Composition

To support the implementation of the IoT4Industry scenario (as explained in Section 2), several base technologies and approaches have been selected to achieve a proof-of-concept implementation. Emerging open-source IoT software components have been analysed and some of them was selected for further implementation. Selected tools are discussed in a separate report on emerging IoT platforms for the Industrial Internet, "*Report on emerging IoT platforms for Industrial Internet*".

To enable a secure and privacy-aware software design, we had to ensure that the used technologies are able to provide (a) transport level security for encrypting data communicated via the Internet, (b) authentication mechanisms to identify the users, and (c) fine-grained authorization models to all data and services. Finally, the selected technology should allow standards-based communication between the involved services, devices and users, to easily interoperate with existing services and applications. This can be achieved by designing RESTful APIs based on the HTTP protocol and using JSON (or XML, if required) as a serialization format. This also enables integration into web applications running in modern browsers on multiple platforms.

For our prototype implementation we use technologies such as the node.js-based "Node-RED" [2] application and the Python-based Micro-framework for web applications called "Flask" [3]. Sensors send their information to Node-RED either by using MQTT or HTTP. Node-RED supports a lot of other messaging interfaces, which can be used in our future applications.

In the following subsections we describe requirements, the underlying printing job distribution and control, the design rules for our REST-API and the technologies currently in use.

3.1 3D Printing Job Distribution and Control

In our scenario, the 3D printers act as units for the production of individualised products ("lot size 1"). To get the customer's job to the printer, several abstraction layers are required. Technically, the 3D model submitted by the customer needs to be analysed, and depending on the properties of the model and additional requirements of the customer (material, colours, quality, etc.) it will be dispatched to a printer that is able to perform this specific job. Before the job started on a particular printer, the model needs to be translated into the machine specific interpretable GCode.

Distribution of printing jobs to a specific printer requires a scheduler, which selects a suitable printer and puts the task into the queue of this printer. Each printer is connected to a printer controller, which sends the machine code to the printer and monitors the status of the printer. This controller provides the IoT-enabled APIs, which are described in the next section.

3.2 REST-API Design and Implementation

To deliver the services for the different user roles required for the 3D printing scenario, we provide a set of REST services that can be accessed by the user interface and other high level services. We therefore consider a uniform approach to be used in the scenario, meaning that sensors, actuators and services should follow a similar communication patterns for remote systems.

General rules in the service API design are sustainability, extensibility, and self-descriptiveness. In addition to being self-descriptive for a human user (developers of higher level services and user interfaces), we also want the API to be self-descriptive for machines to achieve semantic interoperability, which can be later exploited by the software agents by connecting them to ontologies. One possibility to easily semantically annotate IoT services is to use JSON-LD [4], that allows the extension of existing JSON responses by Linked-Data Annotations. This approach was inspired by the works of David Janes for the “Internet of Things Database“ [1]. The main advantage of the semantic annotation is that software agents are able to interpret the results returned by the API, because the context provided in a JSON-LD response contains a reference to a common vocabulary, such as schema.org or dbpedia.org.

In this section we describe the REST-API design of the printer controller, which runs on a Raspberry Pi, as discussed in Section 2.2.2. The printer controller has to manage the printer itself, the model files that are going to be printed, as well as the printing jobs that are submitted to the printer. Therefore the entry point (`http://192.168.48.30:8000/upc/api/v1.0/`) to the REST API of the printer controller (upc = Ultimaker Printer Controller) firstly exposes the services that are available. It provides the following response, which is described below.

Note, that JSON-LD keywords are starting with an “@”-sign.

```
$ curl -i http://192.168.48.30:8000/upc/api/v1.0/
HTTP/1.0 200 OK
Content-Type: application/json
{
  "@base": "http://192.168.48.30:8000/upc/api/v1.0",
  "@context": {
    "services": "https://www.wikidata.org/wiki/Q557770"
  },
  "services": {
    "files": {
      "@id": "http://192.168.48.30:8000/upc/api/v1.0/files",
      "@type": "Collection"
    },
    "printers": {
      "@id": "http://192.168.48.30:8000/upc/api/v1.0/printers",
      "@type": "Collection"
    },
    "tasks": {
      "@id": "http://192.168.48.30:8000/upc/api/v1.0/tasks",
      "@type": "Collection"
    }
  }
}
```

The JSON root node has 3 elements, “@base”, “@context” and “services”:

- “@base” provides the Base IRI[9], which in this case points to the document itself.
- “@context” describes the contextual information of the shorthand object names used in the rest of the JSON-document. In the example, the “services” are linked to the concept of “Web-API” on Wikidata.
- “services” lists the Web-APIs that are provided by this printer controller, which could be: **“files”** to manipulate the files on the controller, **“printers”** to receive information and control the printers that are connected to the controller, and **“tasks”** that show finished, scheduled or current 3d-printing tasks. Each of the services provide an “@id” that links to the entry point of each particular service.

As identified by the service listening on the Base URL, the “Printer” service is located at: `http://192.168.48.30:8000/upc/api/v1.0/printers.`

An example response for calling that service is shown below:

```
$ curl -i http://192.168.48.30:8000/upc/api/v1.0/printers
HTTP/1.0 200 OK
Content-Type: application/json

{
  "@base": "http://192.168.48.30:8000/upc/api/v1.0",
  "@id": "/printers/",
  "@type": "Collection",
```

[9] Internationalized Resource Identifier

```
  "@context": {
    "printers": "https://www.wikidata.org/wiki/Q557770"
  },
  "member": [
    {
      "@context": {
        "name": "http://schema.org/name",
        "url": "http://schema.org/url"
      },
      "@id": "http://192.168.48.30:8000/upc/api/v1.0/printers/um2",
      "@type": "http://dbpedia.org/resource/3D-printer",
      "name": "um2",
      "url": {
        "@id": "http://192.168.48.30:8000/upc/api/v1.0/printers/um2"
      }
    }
  ]
}
```

The response contains members of the printer collection that is controlled by this printer controller. Again, using JSON-LD markups, further semantic information is provided, e.g. that the printer type is a 3D-printer. In addition to the semantic information, the response contains again links to the single printers, in our case just one printer, which is the "Ultimaker 2" already described above. Following this link, further information on the printer is provided as shown below:

```
$ curl -i http://192.168.48.30:8000/upc/api/v1.0/printers/um2
HTTP/1.0 200 OK
Content-Type: application/json

{
  "@base": "http://192.168.48.30:8000",
  "@context": {
    "device": "https://www.wikidata.org/wiki/Q385390",
    "meta": "http://dbpedia.org/resource/Metadata",
    "model": "http://schema.org/ProductModel",
    "name": "http://schema.org/name",
    "port": "https://www.wikidata.org/wiki/Q385390",
    "prn": "https://www.wikidata.org/wiki/Q216601",
    "state":

"https://en.wikipedia.org/wiki/State_%28computer_science%29",
    "usbId": "https://www.wikidata.org/wiki/Property:P1167"
  },
  "@id": "http://192.168.48.30:8000/upc/api/v1.0/printers/um2",
  "@type": "http://dbpedia.org/resource/3D-printer",
  "device": "/dev/ttyACM0",
  "meta": {
    "@id": "http://192.168.48.30:8000/upc/api/v1.0/printers/um2/meta"
  },
  "model": "Ultimaker 2",
  "name": "um2",
  "port": "/dev/ttyACM0",
  "prn": "Printer<id=0x1329c50, open=True>(port='/dev/ttyACM0',
          baudrate=115200, bytesize=8, parity='N', stopbits=1,
          timeout=0.05, xonxoff=False, rtscts=False, dsrdtr=False)",
  "state": {
                                                              "@id":
"http://192.168.48.30:8000/upc/api/v1.0/printers/um2/state"
  },
  "usbId": "USB VID:PID=2341:0010"
}
```

The response contains a “@context” section, which describes what is the semantic meaning of the shorthand labels such as “device”, “meta”, “state”, etc., and also links to further information such as “meta”-information about the printer or the fluctuating “state” information on the printer. Using the “state” information the UI shows what the printer is currently doing, e.g. if the printer is on or off, but also shows what task is currently printed. From the state information, there is a link to the task-service that was already mentioned in the beginning of this section.
To implement the described APIs, the Python-based web-micro-framework Flask [3] has been selected to expose the functionality to internal and external clients. It can be quickly deployed in local prototype scenarios, or scaled up using an external WSGI- or FCGI-capable web-server such the Apache Web Server or Lighttpd.

3.3 Connecting devices using Node-RED

Sensors are not always able to provide extensive service APIs as described above, due to their performance or memory limits. Low-power sensors only provide an analog or digital signal that needs to be transmitted to another device that can interpret the measurement value and enriched it with meta-information. A useful software for this purpose is Node-RED, a tool for wiring together hardware devices, online services and APIs. Node-RED is based on node.js, which itself is based on Chrome’s JavaScript runtime and designed for building fast, scalable network applications. Node-RED also largely fits the described prerequisites of our scenario in

terms of privacy and security. It supports encryption and customizable user authentication to integrate with external authentication infrastructures.
It allows consuming data from various existing devices or services (including REST-APIs, MQTT notifications, etc.) and exposing own new APIs to external clients according to the previously described API design. It therefore allows us to rapidly prototype connections between the components of our scenario.
Figure 11 provides an example of a data flow constructed with Node-RED that collects MQTT-messages, processes them and saves to a local file. Furthermore, HTTP-POSTs are generated for an external web services, and in addition, a secure websocket endpoint is provided to collect the data from an UI.

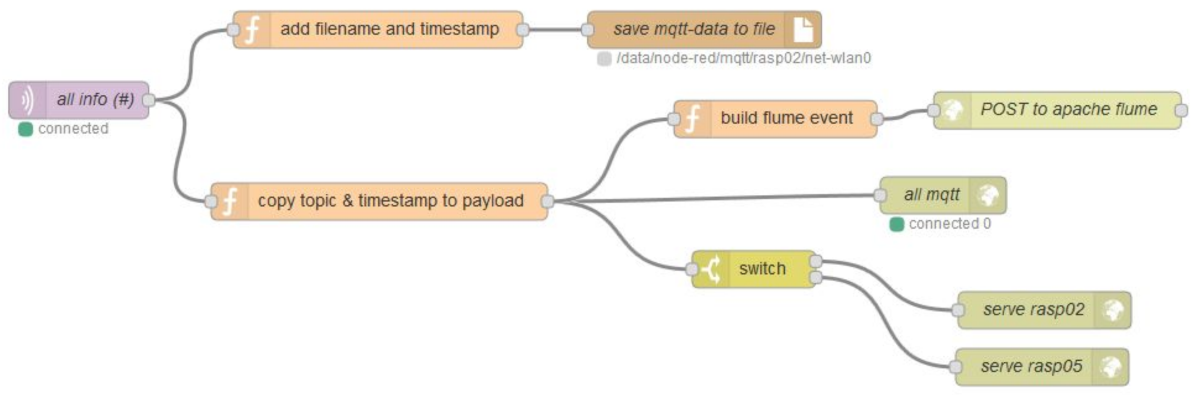


Figure 11. Node-RED data streaming flow

4. Publication

An interactive demo version of the presented scenario and technologies has been published and presented at the 15th anniversary of the Salzburg Research:
https://demo.industrielles-internet.at/v1/
Using the following login data, a version with static and recorded data can be accessed:

Login:	demo
Password:	pass123

Furthermore the code base is available on our internal Bitbucket GIT and all components are documented on our internal Confluence Wiki.

5. Conclusion

The IoT offers great opportunities for enhancing production processes to an advanced level. Still the existing tools and technologies can be used to embrace those opportunities without giving up security, privacy or data control, but we will need to change the way how data is currently handled.

The scenario of the 3D printing farm can easily be conceptualized and adapted to other use cases without changing the underlying architectural approach to a great extent. For example, manufacturing plants could implement simple REST APIs to integrate their production cells in an agent-based architecture. Besides that, the usage of existing technologies and open source solutions and the distribution on numerous agents allow for employing inexpensive replaceable endpoints, which can than serve as a scalable foundation for industrial or other IoT/WoT applications.

Furthermore, the involvement of customers and other enterprises would establish new opportunities in terms of cooperation, exploitation of usage data and continuous product improvement.

References

[1] IOTDB: The Internet of Things Database, https://iotdb.org/

[2] Node-RED: A visual tool for wiring the Internet of Things, http://nodered.org

[3] Flask: A Python Microframework, http://flask.pocoo.org

[4] JSON-LD: http://www.w3.org/TR/json-ld/