AutoIoT: A Framework based on User-driven MDE for Generating IoT Applications

AUTOIOT

Fraunhofer

by Thiago Nepomuceno

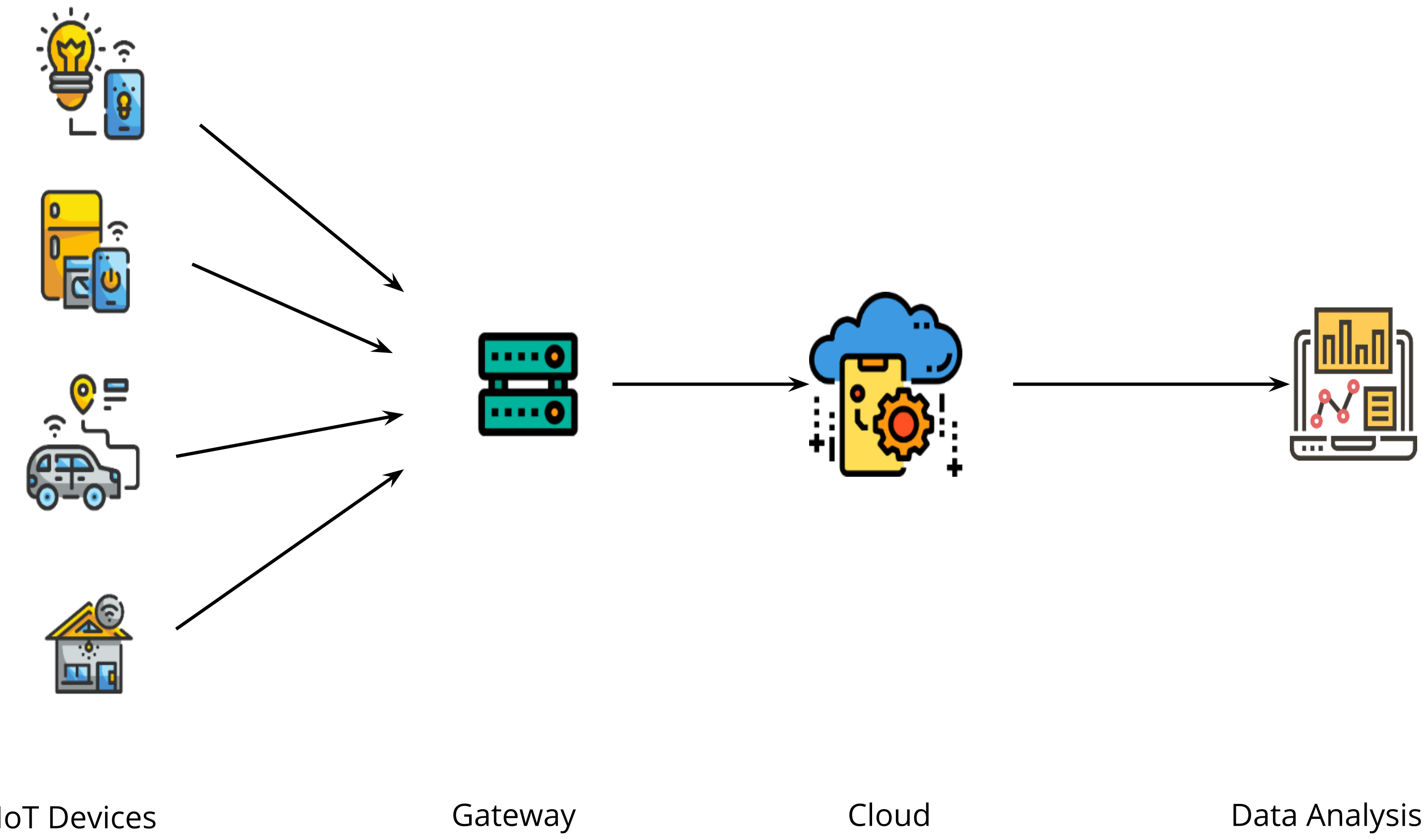
The 35th ACM/SIGAPP Symposium On Applied Computing

Agenda

Fraunhofer IIS

- Fraunhofer IIS is the biggest Fraunhofer institute. Famous for creating the MP3 standard.
- Among other areas, it also does research in the **Internet of Things**.
- Being more specific I work at the department **Data Spaces and IoT Solutions**.
- We have experience doing **many research and industrial projects** in the IoT field.

Example of IoT System



https://www.flaticon.com; authors: eucalyp and wanicon.

IoT Cloud Server: Basic Features

- Receive sensor data from the gateway.
- Store the data in a database.
- Basic visualization of the data (ideally both historical and real-time).
- Share the stored data with third-party systems (e.g. an Web API).
- Device Abstraction: allowing thid-party system to communicate with devices using a high-level interface (e.g. an Web API).

IoT Cloud Server Development: IoT Platforms

Google Cloud

aws

Microsoft Azure

Bosch IoT Suite

IoT Cloud Server Development: IoT Platforms

Generally a good choice for most companies. However, as a research institute that works with many different partners from different fields. We faced the following problems:

- Vendor lock-in.
- Privacy.

IoT Cloud Server Development: Alternatives

Node-RED

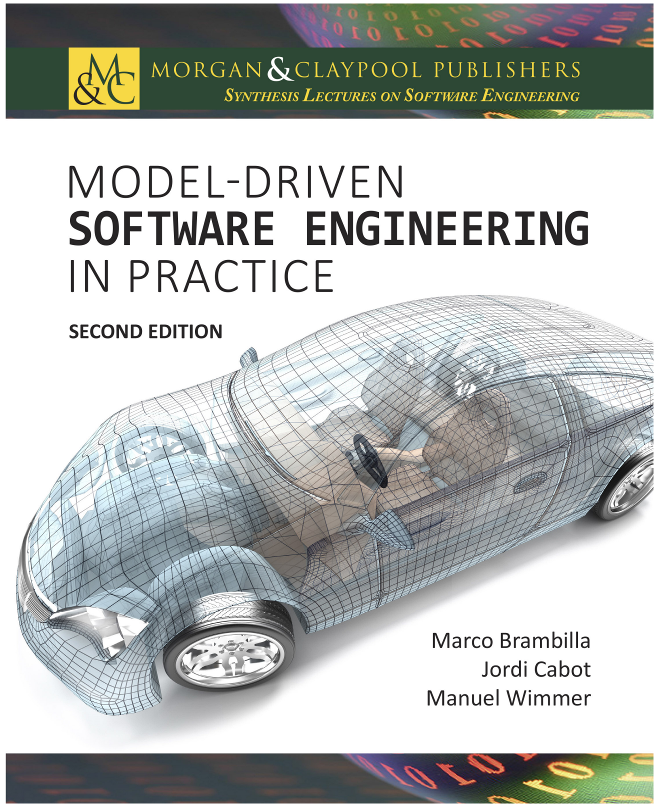
open source

IoT Cloud Server Development: Alternatives

Mostly of the existing alternatives had one or more of the following problems:

- Too much effort learning.
- Did not really increase productivity.
- Too much effort maintaining.
- Not suitable for bigger scenarios.

IoT Cloud Server Development: Model-Driven Engineering


MORGAN & CLAYPOOL PUBLISHERS

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

MODEL-DRIVEN

SOFTWARE ENGINEERING

IN PRACTICE

SECOND EDITION

Marco Brambilla

Jordi Cabot

Manuel Wimmer

IoT Cloud Server Development: Model-Driven Engineering

- Focused in models and modeling.
- Most approaches require great expertise in MDE and/or modeling.
- Most approaches only generated the source code partially (no executable).
- Hard to convice a team used to do standard development (focused around the source code) to try something new.

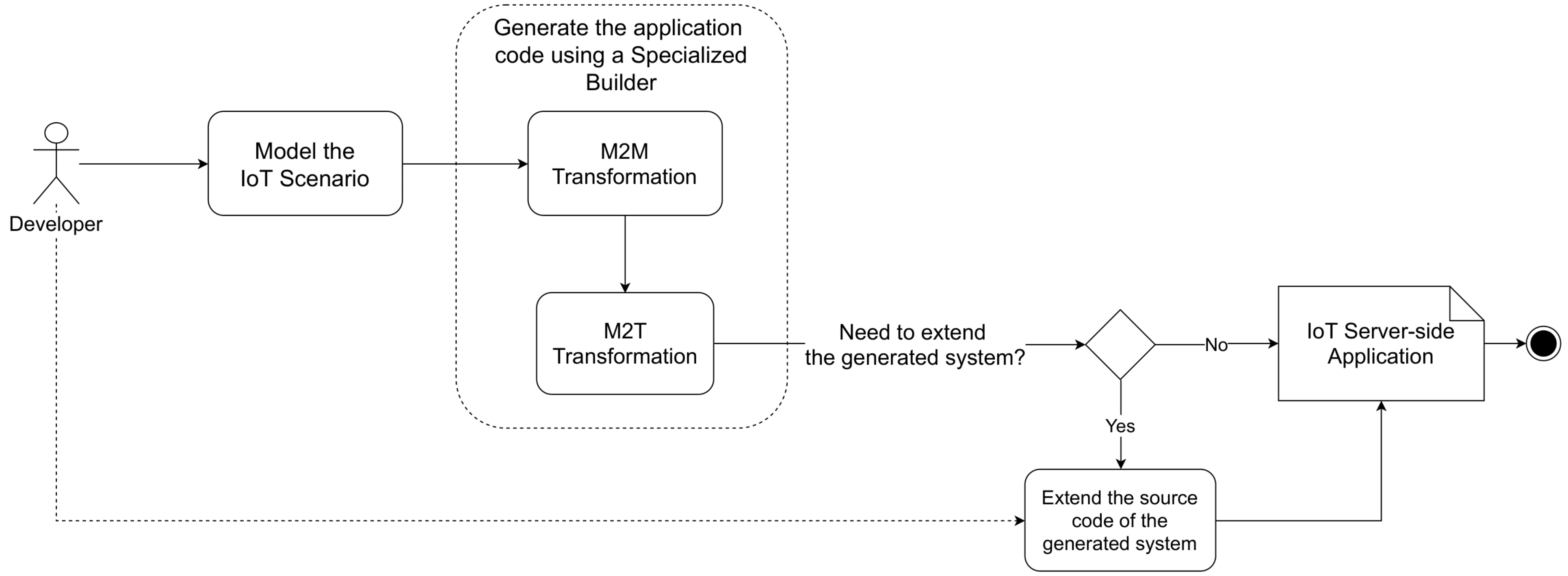
IoT Cloud Server Development: Small Development Team

After try many alternatives, the development team got the experience and felt confident to discuss what have worked and what have not. The team wanted a tool that:

- Gives them higher productivity (more projects done in less time).
- Is reusable in different projects.
- Is easy to learn and use.
- That create a system easy to deploy.

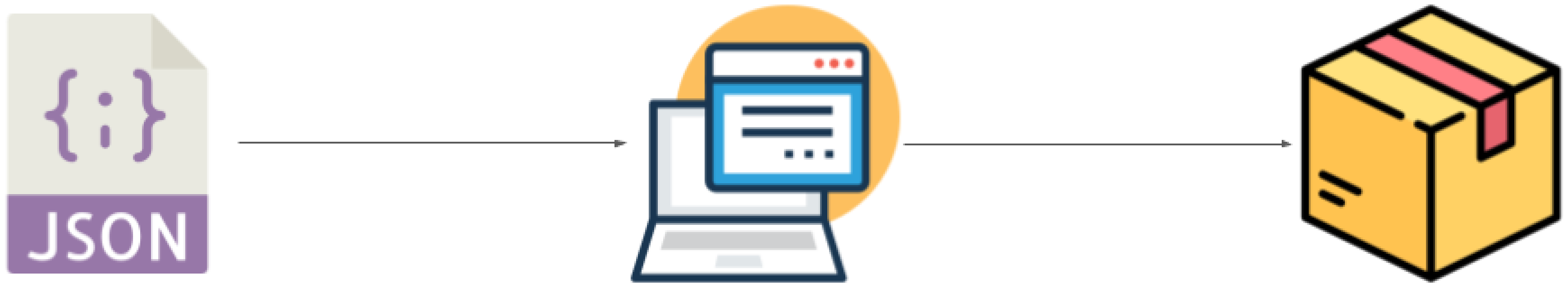
AutoIoT

Internally, the process workflow is similar to other MDE approaches.


Generate the application code using a Specialized Builder
Developer
Model the IoT Scenario
M2M Transformation
M2T Transformation
Need to extend the generated system?
No
Yes
IoT Server-side Application
Extend the source code of the generated system

AutoIoT

From the developer point-of-view it is a lot simpler than usual MDE approaches.



AutoIoT: Project Representation in JSON

```json
{
  "project": {
    "name": "Container Management Project",
    "description": "The Container Management Project.",
    "app_port": 5000
  },
  "database": {
    "type": "sqlite",
    "hostname": "localhost"
  },
  "mqtt": {
    "hostname": "iot.eclipse.org",
    "port": 1883
  },
  "devices": [
    {
      "name": "Container",
      "description": "An IoT device that sends temperature, position and filing status.",
      "fields": {
        "name": "String",
        "barcode": "String"
      },
      "sensors": [
        {
          "name": "MainSensor",
          "fields": {
            "send_interval": "String"
          },
          "data_fields": {
            "temperature": "Float",
            "filing_status": "Integer",
            "position": "Point"
          }
        }
      ]
    }
  ]
}
```

AutoIoT: Generating the Project using the Python Library

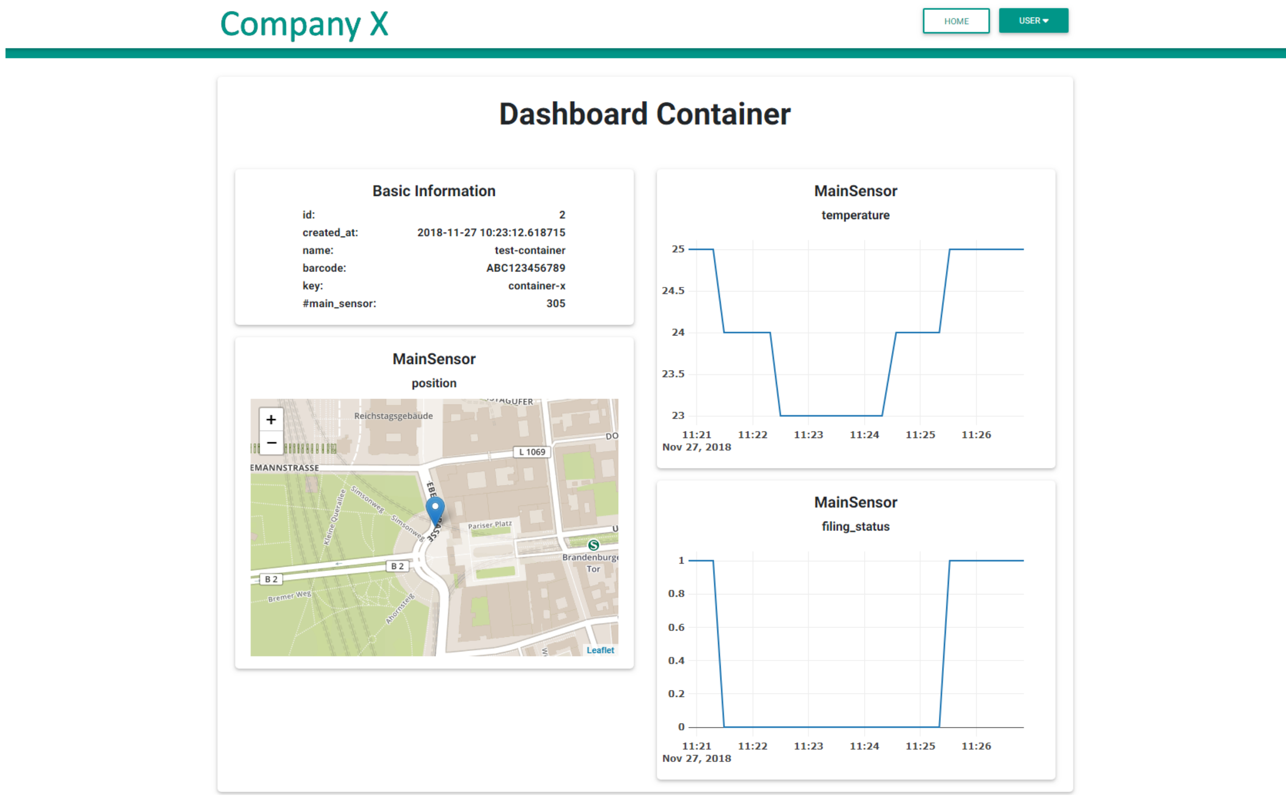
```python
from autoiot import AutoIoT
from autoiot.builders import PrototypeBuilder

autoiot = AutoIoT()
autoiot.load_project('project.json')
autoiot.build(PrototypeBuilder, 'output/project', {'docker': True})
```

AutoIoT: Generated Software

The generated source code is ready to deploy as it is or can be modified to incorporate additional functionalities.

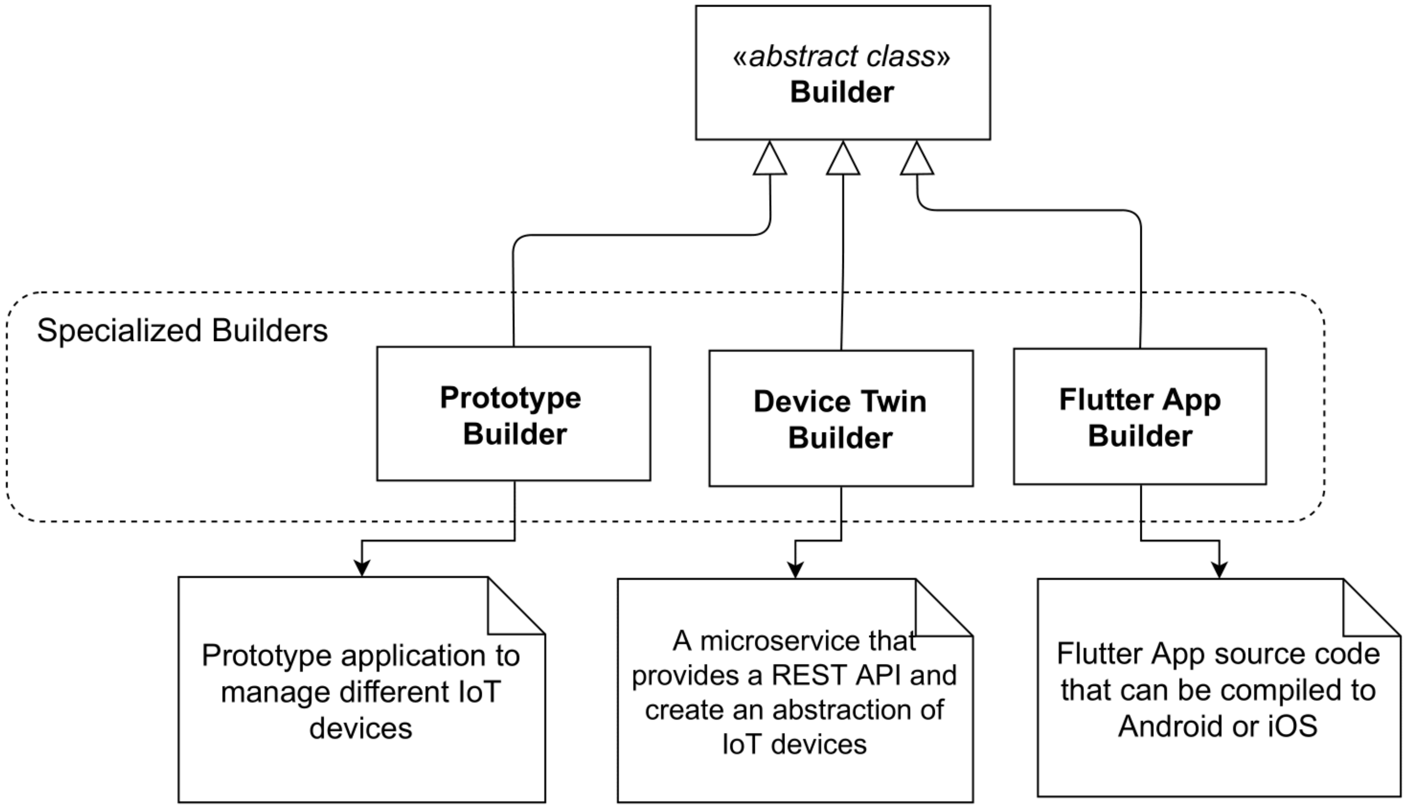
- Receive sensor data from the gateway.
- Store the data in a database.
- Basic visualization of the data (ideally both historical and real-time).
- Share the stored data with third-party systems (e.g. an Web API).
- Device Abstraction: allowing thid-party system to communicate with devices using a high-level interface (e.g. an Web API).



Basic Information

id:	2
created_at:	2018-11-27 10:23:12.618715
name:	test-container
barcode:	ABC123456789
key:	container-x
#main_sensor:	305

AutoIoT: Different Types of Output


«abstract class»
Builder
Specialized Builders
Prototype Builder
Device Twin Builder
Flutter App Builder
Prototype application to manage different IoT devices
A microservice that provides a REST API and create an abstraction of IoT devices
Flutter App source code that can be compiled to Android or iOS

AutoIoT: Evaluation

The AutoIoT Framework has been applied to speed-up development of projects inside **Fraunhofer IIS** during the past few years.

During the writting of this paper we invited external developers to try it and give they honest feedback.

Partipants had a description of an simple IoT scenario (a smart box sending sensor data to a server) and were asked to use AutoIoT to generate a system to the given scenario. Additionaly they were asked about their expertise in some technologies.

In total 52 people mixed from academy and industry participated in the experiment.

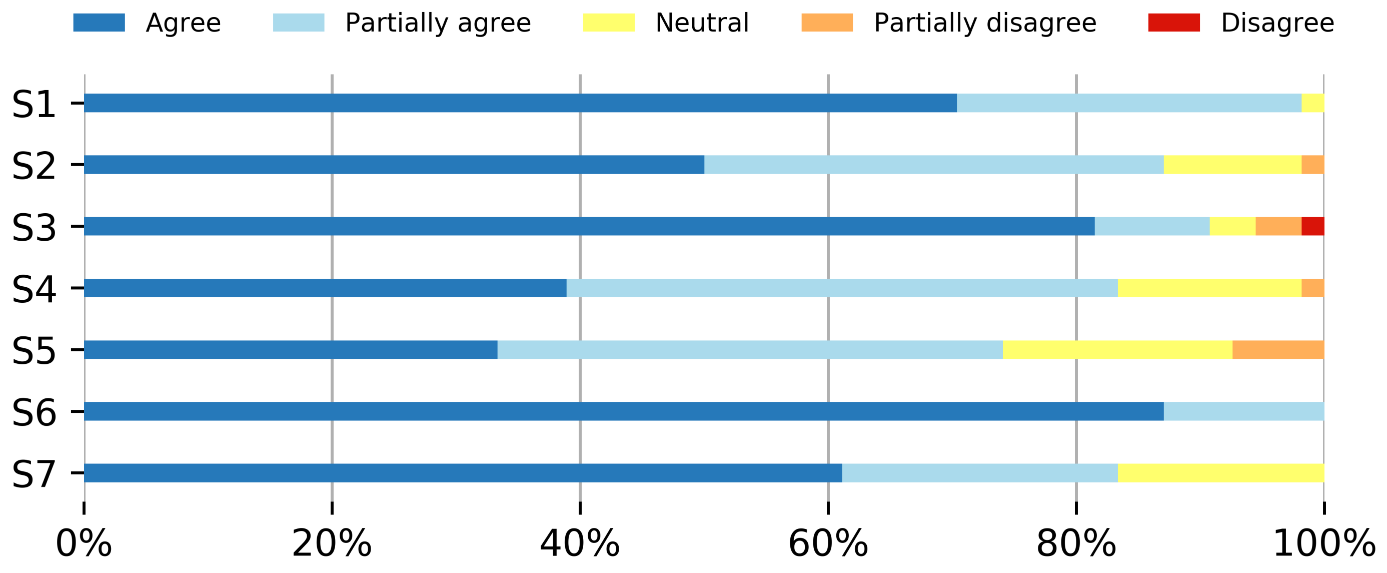
AutoIoT: Evaluation

After using AutoIoT to generate the system to the IoT scenario, the participants were asked if they agree or not with the following statements.

- S1 - The system generated by AutoIoT covers the requirements of the scenario.
- S2 - After reading the project description file, I could easily change the configuration to cover a scenario with different IoT devices.
- S3 - I would spend more than 40 working hours to manually codify the same system generated by AutoIoT.
- S4 - The code generated by AutoIoT is well organized and easy to understand.
- S5 - If I had to change the generated code and change the way that the sensor data is handled, I would know which file I should change.
- S6 - Use AutoIoT to generate the server application was easy.
- S7 - I could use AutoIoT to generate the server-side application in some of my IoT projects in the future.

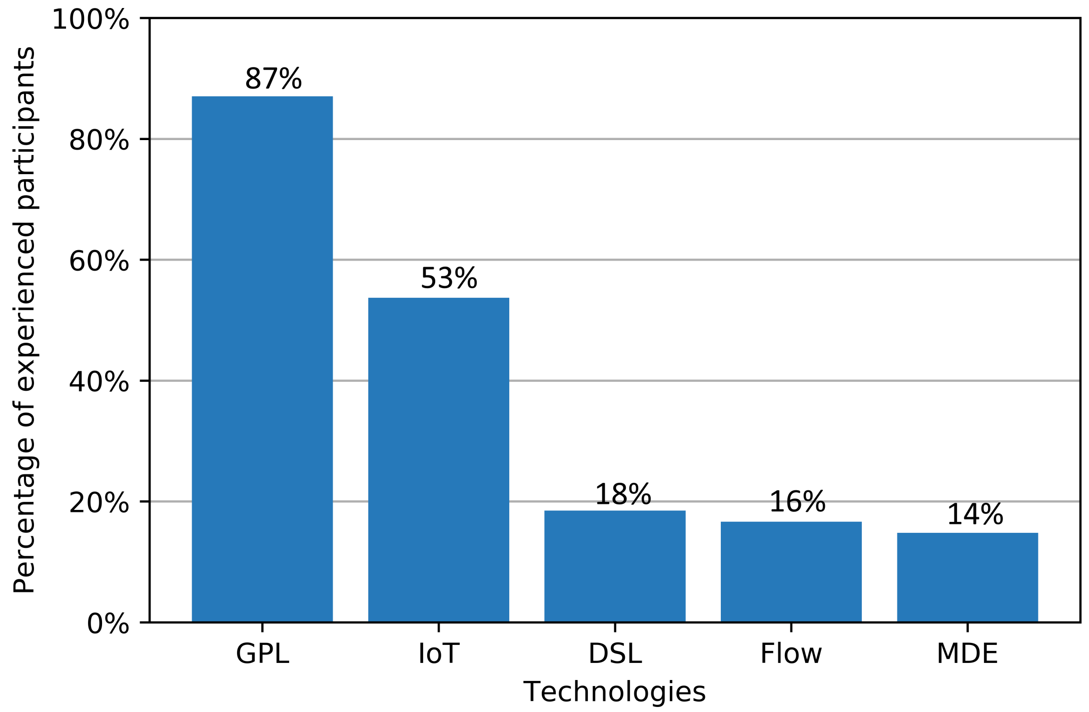
AutoIoT: Evaluation

As we can see below, most of the participant *Agreed* or *Partially Agreed* with the statements.



AutoIoT: Evaluation

When asked about their expertise in some technologies, few of the participants had knowledge in *Domain Specific Languages*, *Flow-Based Programming*, and *Model-Driven Engineering*. However, almost everyone declared to have expertise in *General Programming Language* and *IoT*.



AutoIoT: Conclusion

- According to the experiment, the participants **agreed** that AutoIoT is efficient when used to generate code for IoT scenarios.
- AutoIoT only uses technology that **most developers** are already **familiar** with, making it easier to convince developers to try the framework.
- AutoIoT was already used to speed-up the development in multiple projects inside **Fraunhofer IIS**. However, in its current status, it is mostly used to speed-up prototype development.
- Future works should incorporate and evaluate different generators that can be used in bigger/more complex IoT projects.

Thanks

Bavarian Ministry of Economic Affairs, Regional Development and Energy

Fraunhofer IIS

ADA LOVELACE CENTER