# Low-Code Internet of Things Application Development for Edge Analytics

Hafiz Chaudhary, Ivan Guevara, Jobish John, Amandeep Singh, Tiziana Margaria, Dirk Pesch

# Low-code Internet of Things Application Development for Edge Analytics

Hafiz Ahmad Awais Chaudhary[1,4], Ivan Guevara[1,4], Jobish John[2,4], Amandeep Singh[1,5], Tiziana Margaria[1,3,4,5], and Dirk Pesch[2,4]

[1] University of Limerick, Ireland
[2] University College Cork, Ireland
[3] Lero - the Software Research Centre,
[4] Confirm Centre for Smart Manufacturing, Ireland
[5] Centre for Research Training in Artificial Intelligence (CRT AI)
{ahmad.chaudhary,ivan.guevara,amandeep.singh,tiziana.margaria}@ul.ie
{j.john, d.pesch}@cs.ucc.ie

**Abstract.** Internet of Things (IoT) applications combined with edge analytics are increasingly developed and deployed across a wide range of industries by engineers who are non-expert software developers. In order to enable them to build such IoT applications, we apply low-code technologies in this case study based on Model Driven Development. We use two different frameworks: DIME for the application design and implementation of IoT and edge aspects as well as analytics in R, and Pyrus for data analytics in Python, demonstrating how such engineers can build innovative IoT applications without having the full coding expertise. With this approach, we develop an application that connects a range of heterogeneous technologies: sensors through the EdgeX middleware platform with data analytics and web based configuration applications. The connection to data analytics pipelines can provide various kinds of information to the application users. Our innovative development approach has the potential to simplify the development and deployment of such applications in industry.

**Keywords:** Low Code · Model Driven Development · Edge Analytics

## 1 Introduction

The Internet of Things (IoT) enables the communication between integrated infrastructure of connected sensors, devices and systems with the aim to provide innovative solutions to various data acquisition and decision challenges across different domains. Research [14] has shown a substantial rise in adoption of a wide range of IoT devices and platforms over the past few years. 29.4 billion IoT devices are expected by 2030 according to [31], a twofold increase compared to the expected number at the end of this year. There is a growing necessity for these different devices and solutions from different service providers to seamlessly work together, as each vendor provides support for its own devices and infrastructure, leading to interoperability issues in the overall ecosystem.

Software development cycles, in the context of IoT applications, are becoming more complex as the challenge for interoperability increases. The complexity across several functional layers composing novel IoT architectures requires deep technical knowledge and cross-functional integration skills for each bespoke system. The learning curve presented by this kind of heterogeneous system development demands simplification of their design, construction, and maintenance. IoT middleware platforms have become an essential part of the IoT ecosystem, as they provide a common interface between the different sensors, computing devices, and actuators. As shown in Ali et al. [3] these platforms fulfil different types of requirements, which leads adopters to perform an exhaustive analysis before choosing and implementing a specific type of architecture. From the perspective of application developers, incompatibility between IoT platforms results in adapting their application to the platform-specific API and information models of each different platform, which makes cross-platform development harder and time-consuming.

We present a low code approach designed to simplify the development of IoT applications. For this we adopt EdgeX Foundry [9] as our example integration platform as it is a well-known, highly flexible open-source software framework addressing the challenge of interoperability between a heterogeneous set of devices, protocols and IoT objects. EdgeX provides a way to homogenize the data sent from different protocols, providing a single data structure that facilitates the way we retrieve information from the Edge. The framework is structured in different layers, each composed of multiple microservices.

In the context of smart manufacturing, we are working on a Digital Thread platform[18] based on model driven development, that provides automatic code generation and deployment of heterogeneous applications that require the interoperability across different systems, technologies and programming paradigms. We choose model types with formal semantics, so that they are amenable to formal verification and analysis. This choice is due to the fact that we believe in the benefits of early validation and verification at the model level and automated support for syntactic and semantic correctness.

In this study, we work with the low/no-code development environments DIME [5] and Pyrus [35], and follow the native library approach [6] to extend the range of systems and domain-specific functionalities that they provide. Fig. 1 shows the current architecture of the Digital Thread platform, where the modelling layer provides the essential application modelling capabilities e.g. GUI, data persistence etc. The Process layer, in combination with already implemented common DSLs and External Native DSLs, models the business logic to the application. The integration of different frameworks and technologies is encapsulated in the External Native DSLs and orchestrated in the Process models.

In the following, Sect. 2 summarizes related work, Sect. 3 gives an overview of the software and hardware technologies and tools adopted for the use case, Sect. 4 details the architecture and the new functionalities in the system, Sect

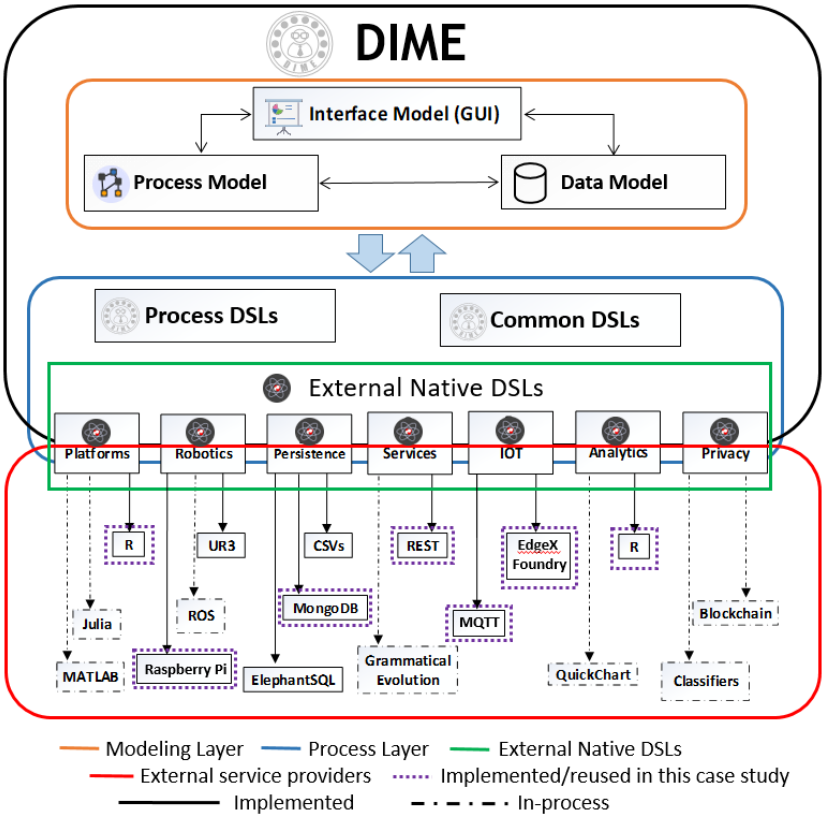



Fig. 1: Architecture Overview of the Digital Thread Platform

5 describes the application development and the processes, Sect. 6 discusses the results, and Sect. 7 concludes and sketches future work.

## 2 Related Work

Low-code platforms are becoming a key technology in the IT industry. They enable a more straightforward workflow to generate highly complex solutions, hiding implementation details and relying on a model-like paradigm, putting the focus on composing behaviours instead of boilerplate code. Many benefits and opportunities are provided by these types of technologies. It is estimated that the global low-code platform market revenue will reach approximately 65 billion U.S. dollars in 2027, having generated 13 billion U.S. dollars in 2020 [30]. This clearly demonstrates the capabilities and potential of these technologies in the context of a competitive and dynamic market. Despite the predominant position of programming languages and frameworks, several solutions can be found nowadays with this kind of approach: next to CINCO/DIME [24, 5],

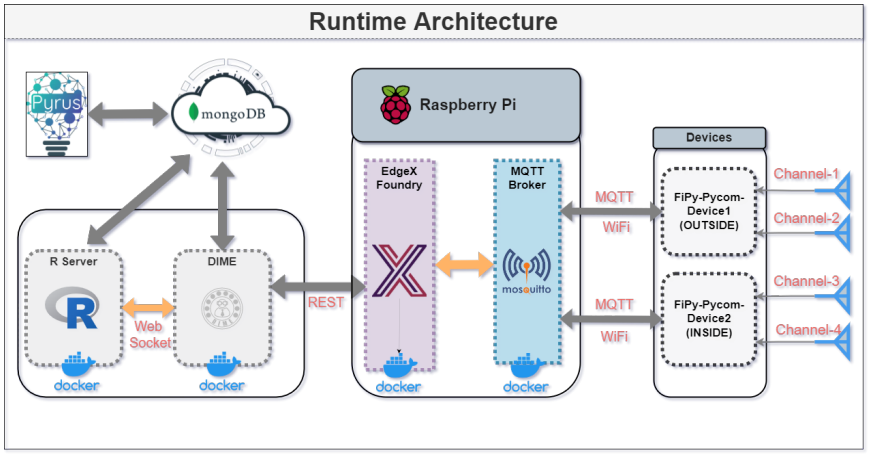



Fig. 2: The Runtime architecture: infrastructure and communications

which have a formal model at their core, several industrial solutions are coming forward, among which Tines [10] specializing in security workflow automation, PTC ThingWorx [25], AWS IoT [4] and Microsoft Azure IOT Suite [23] specializing in IoT and Industrial IoT, H2o.ai [13] for general purpose AI and ML, and Siemens MindSphere [28] for industrial IoT as a Service solutions, that uses analytics and AI to power IoT solutions from the edge to the cloud.

The industrial solutions do not have so far formal models, but they resort instead to the combination of the visual composition of the orchestrations, which is more intuitive than code, with traditional debugging and testing. This can be onerous if one is more remote from the code that is actually running, and has no access to it. Several approaches support dataflow models or control-flow models, but not both. Our choice of models combines the visual appeal of the block-based composition with formal semantics [20, 19], and we carry out the data analytics twice: with DIME (control and data flow) and Pyrus (only dataflow).

## 3 Overview of tools and technologies

Fig. 2 shows the runtime architecture of the end-to-end application under consideration. Accordingly, in this section, we briefly present the software and hardware platforms that it encompasses.

### 3.1 Software platforms

In the following, we present a brief overview of the different software platforms we have integrated to create our low code IoT application development workflow.

**EdgeX Foundry** EdgeX Foundry [9] is one of the Industrial IoT middleware platforms that is being widely adopted by industry, especially for industrial automation applications to exploit the benefits of edge compute and edge intelligence. EdgeX Foundry is a highly flexible hardware-agnostic platform consisting of loosely coupled microservices. The docker deployable microservices required to handle a particular application can be broadly categorised into four service layers. The device service layer provides the required connectors used to provision and connects the external hardware devices/objects such as sensors, actuators, machines, cameras etc. These connectors translate the data coming from a wide variety of devices, each possibly using different protocols, into a common data structure/format that is understood by the microservices in all the different layers. The core services layer holds the most knowledge about the EdgeX system deployment, such as which things are connected, how they are configured, what data is flowing through them, etc. The supporting service layer consists of microservices that provide services such as alerts, edge analytics, scheduling, data clean up etc. The application service layer provides a means to export/send the data from EdgeX to other external software applications such as on-premise applications or cloud platforms like AWS IoT or Google IoT for additional processing. A detailed description of all the layers and associated microservices can be found in some of our prior work [15][16]. While we have chosen EdgeX for this work, similar IoT middleware or integration platforms offering a microservice architecture with REST interfaces would be similarly suitable to our approach described in this paper.

**DIME** DIME[5] is an Eclipse based programming-less graphical modelling environment for prototype driven web application development. It follows the OTA (One Thing Approach)[20] and the XMDD (eXtreme Model-Driven Design)[22] paradigms to modelling and development, and it empowers domain experts to model an end-to-end web application with little or no programming experience. To cover the different aspect of web application, DIME provides a family of native DSLs that provide collections of ready to use functionalities, and it supports the development of new applications via different model types that refer to each other and whose consistency is checked largely automatically. GUI models are structural descriptions of UIs that use basic UI components with data bindings. GUI models cover a range of various basic to complex UI elements, that ease the development of user interfaces that are highly customizable at runtime. Data Models cover the persistency needs of applications, on the basis of UML-type structures. In addition to built-in primitive data types, users can define complex and enum data types, and perform aggregation on attributes and associations. Process models express the most often hierarchical business logic, and detail both the data and control flow. Finally, a DIME application descriptor (DAD) model specifies the needed artefacts for an application, including relevant domain models and an interaction process that serves as the landing page for the project. We use DIME for the development of new applications, that we model with the help of natively supported and newly developed DSLs.

**MongoDB Database** The Atlas NoSQL cloud database by MongoDB offers an optimised solution for JSON-like optional schemas [1]. It natively supports complex data objects occurring in IoT architectures, like time series data. Atlas can handle operational data in real-time with seamless accessibility across heterogeneous systems and stakeholders. This capability addresses a core need of IoT applications, hence we use MongoDB to store the data collected from the Pycom FiPy light sensors. The communication between the application and the cloud instance of the database is implemented in DIME as a MongoDB-specific DSL.

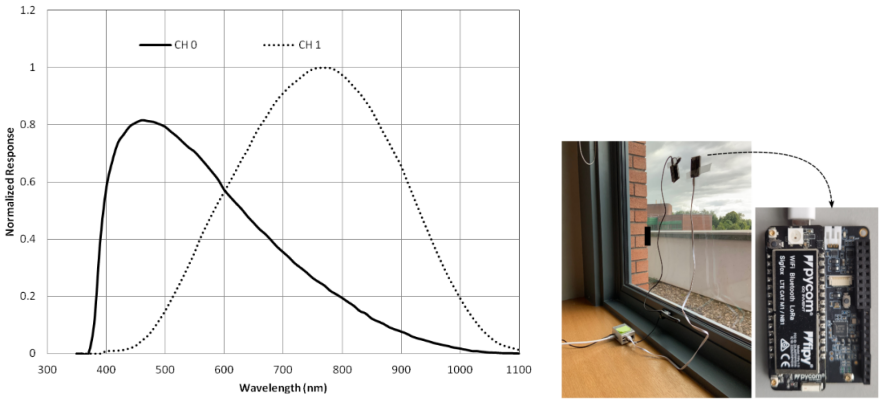
**R Infrastructure** R is a free programming language specifically for statistical and numerical computations and data visualization [33]. We use the *R-serve* package to provide support to the R language in DIME using the TCP/IP protocol [29]. We use R to analyse and visualize the time-series data from the FiPy light sensors. Several packages are imported in the R environment to import data from MongoDB, analyse the time series, and then visualize the data.

**Docker** Docker is an open software platform successfully used to deploy applications as *containers*. Using Docker, applications can be built, tested, deployed and scaled into many environments because it includes support for libraries, system tools, code, and various runtimes [8]. We use Docker here to securely deploy frameworks, services and platforms in separate containers. These containers communicate with each other via different protocols such as MQTT, REST, etc.

**Pyrus** Pyrus [35] is a web based special purpose graphical modelling tool for Data Analytics. It bridges the gap between Python-based established platforms like Jupyter [2] and workflows in a data-flow driven fashion. The single Python functions are implemented and stored in Jupyter, special signature annotations are added to these functions, so that the functions can be identified by the Pyrus web-based orchestration tool, where the pipelines are composed. From the pipelines, Pyrus generates the Python code for the orchestration and configuration, which is again stored and executed in Jupyter. This separation of concerns decouples the coding and development of the single functionalities from the data analytics orchestration modelling, which happens in accordance with model driven engineering principles [21].

### 3.2 Hardware platforms

Here, we discuss the various hardware components that are used for the system implementation. We run EdgeX Foundry on the well-known, and widely used single-board computer, the Raspberry Pi (RPi). We use the 4th generation model B RPi boards with 4GB memory running Ubuntu server as the operating system (OS). There is no particular reason behind our choice of hardware/OS




(a) Normalized spectral response of the light sensor [17]

(b) Experimental set up installation

Fig. 3: MQTT based Pycom IoT module

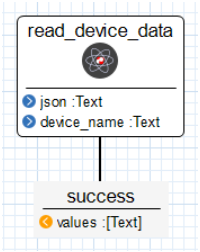
platform other than simplicity, low cost and open-source nature. Any other suitable hardware or OS platform that supports docker based microservices can be used based on specific application demands.

The external "thing" or IoT device we use in our evaluation is the "FiPy" [26] wireless module by Pycom. The device is mounted on a "Pysense" [27] expansion board, also from Pycom. FiPy is a micro python enabled ESP32 microcontroller-based IoT module that supports several wireless connectivity options, including WiFi. We refer to the FiPy module and the Pysense board as "pycom device" in the discussions below. The pycom device acts as an external IoT device that communicates with the EdgeX middleware platform over the MQTT protocol. The provisioning of the pycom MQTT device to EdgeX is done using its device profile, which is a YAML file following the similar procedure as detailed in one of our prior works [16]. The pycom device has different sensors, but we use the light sensor(LTR329ALS01) as an example in this work. The LTR329ALS01 [17] is a dual light sensor that provides digital outputs for external light levels. The sensor periodically reports the light detected at two different wavelengths (blue, red) as Channel 0 and Channel 1 outputs, and the spectral response is shown in Fig. 3a. Thus, we consider the pycom device as an MQTT device having two resources, and we name them as light_ch0 and light_ch1. The remaining sensors may be added as additional resources in the device. However, the approach is the same, hence, we do not include further details in this work.

## 4 Architectural View: the System of Systems

### 4.1 The system architecture

As mentioned earlier, Fig. 2 shows the runtime infrastructure of the end-to-end application under development. Here, two pycom devices with two light channels




(a) SIB representation in DIME

```java
public static List<String> read_device_data(String json, String device_name) {
    List<String> mylist = new ArrayList<String>();

    ObjectMapper objMapper = new ObjectMapper();
    try {
        EdgeX_DB[] myObjects = objMapper.readValue(json, EdgeX_DB[].class);

        for (EdgeX_DB edgeX_DB : myObjects) {

            {
            if(edgeX_DB.name.equals(device_name))
            {
                mylist.add(objMapper.writeValueAsString(edgeX_DB).toString());
            }}
        }
    } catch (JsonMappingException e) {
        e.printStackTrace();
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
    return mylist;
}
```

(b) Java implementation of the body

Fig. 4: read_device_data SIB in the EdgeX collection

each (right) capture the light in different environments. They send that data via MQTT protocol to a MQTT broker running on a Raspberry Pi (middle). The EdgeX Foundry listener reads this data from the MQTT broker and stores it in a local schema.

On the left side, the low-code Web application developed using the DIME model driven development framework runs in a Docker container. The web application initiates the communication with the EdgeX Foundry over a REST protocol, it reads the device data from the local schema of the EdgeX Foundry and does further processing. In this case, it stores the data into a MongoDB NoSQL database, instructs the R server to perform analytics computations on this data, and then sends back the results to the Web application for their visualization and presentation to the users in a graphical format.

Altogether, we see here a variety of systems and subsystems, spanning various hardware and software platforms, protocols, runtimes, and programming languages, that are successfully orchestrated to produce a visualization, an understanding and an interpretation of the data, in two different ways. The key points of this low-code approach are

- the ease of producing virtualized representatives of the disparate system's capabilities in the MDD environments: following the "write once" principle, the Native Service Independent Building Blocks(SIBs) delivers this convenient abstraction, followed by the
- reusability of these ready-made SIBs in many applications, potentially across application domains.

### 4.2 Integration of the Heterogeneous Subsystems: The Native SIBs

Fig. 4a shows how a typical SIB looks like when it is used in a process model. This specific SIB is the read_device_data SIB that belongs to the EDGEX.sibs

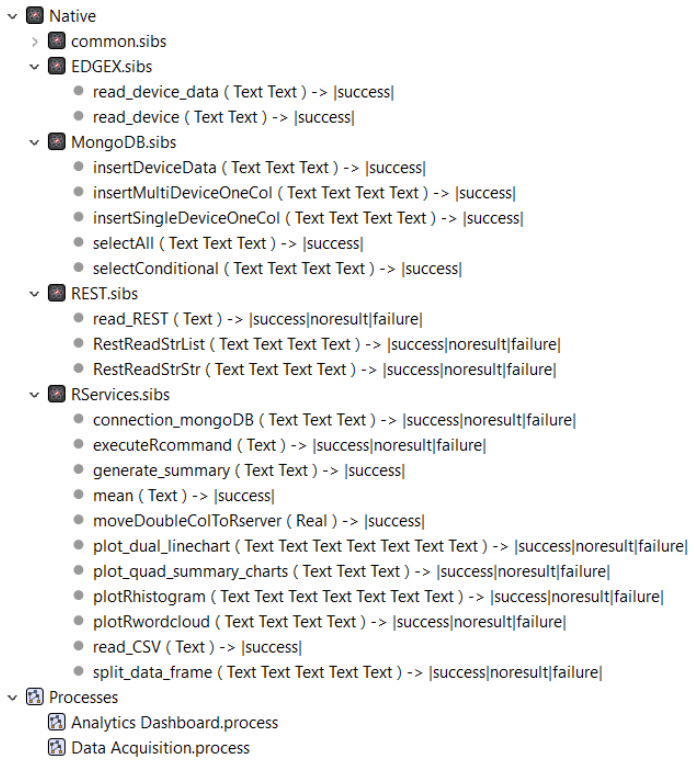



Fig. 5: Native SIBs Collections (SIB palette developed) and Processes in DIME

palette shown in Fig. 5. From this visual representation we see that it requires two inputs, json and device_name, that are fed to this block as either a static value or from another component, as shown via the data flow arrows. The body of the SIB is shown in Fig. 4b. At runtime, this backend functionality is invoked, and upon successful execution the outgoing control branch labelled success is taken, with the corresponding data output to be used later in the workflow. If there are runtime issues, an error branch is followed instead. Every SIB has an error branch to deal with exceptions. The error branches typically output messages and/or take corrective/mitigation actions. They are not shown in this picture nor in the processes, where we concentrate on the normal behaviour. Extending the capabilities of the platform through this integration when developing a new application is the low-code part of the development process: only missing functionalities need to be added, and this happens in a local, small scale development that uses the programming language and runtime of the target system.

### 4.3 No-code Reuse through the Native SIB Palettes

In Fig. 5 we see the collections of new SIBs developed for this application, categorised in their respective Native DSL. This way it is easy to find the Native

DSL to access EdgeX devices, MongoDB, the R platform, REST services etc., that grow with the growth of the platform. Every SIB is developed once, made available to the platform, for further reuse. This is the ease of no-code reuse, so that over time the development of applications that involve systems for which the integration is already available becomes increasingly a no-code task, with low-code new integrations only required for new systems, or when new functionalities for existing systems are added.

## 5 Application Development: the Processes

At the bottom of Fig. 5 we see the list of processes that have been developed for this application: the Data Acquisition process and the Analytics Dashboard process. We describe them in detail, together with an alternative implementation of the analytics in Pyrus.

### 5.1 Data Acquisition from IoT devices

Fig. 6 shows the business logic of the web application for the data acquisition from the IoT devices and ingestion into MongoDB. Our fully functional process is rather simple and typical of such applications. It is intended to be used as a demonstrator, as a blueprint for subsequent applications, as well as for application development training purposes.

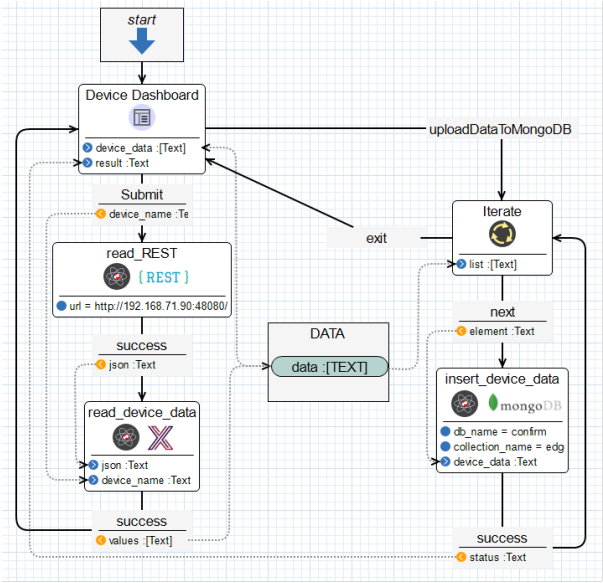



Fig. 6: Data Acquisition from EdgeX Foundry and Data Ingestion into MongoDB - Process in DIME

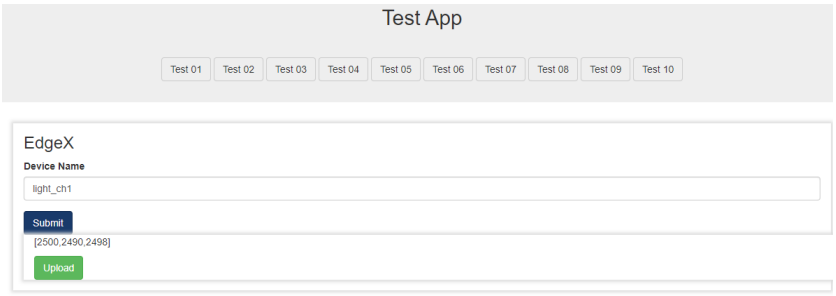



Fig. 7: IoT Web Application: the Device Name Input GUI

The start SIB indicates that we have here a stand-alone process, that does not receive any inputs from the context. The application displays the Device Dashboard shown in Fig. 7, implemented through a GUI model, where a user writes the name of the IoT device of interest. On clicking submit on the GUI, the control flow moves to successive SIB read_REST that has the server url as static value. This SIB reads the device data from the EdgeX Foundry instance running on the Raspberry Pi, and retrieves the corresponding JSON. The SIB read_device_data, the SIB we presented in Sect. 4.2, parses the JSON received from EdgeX Foundry and extracts the device related instances, which are passed as input to this SIB from the GUL SIB. We see here that the dataflow and the control flow differ: while so far we have a linear pipeline in the control flow, the data for some SIBs is provided by various components, at various times. It is therefore useful for the developer to be able to see and design or check separately both the control flow (the logic of what is done, step by step) and the flow of data, which is typically prepared in a number of steps and then consumed by a SIB that collects a number of inputs and processes them.
At this point, the control flow and dataflow return to the GUI, where the extracted data is displayed on the web page. If the user decides to store this data, by clicking on the Upload button, the control flow proceeds to the right side of the Process in Fig. 6: the iterate goes through all the tuples of the dataset and inserts them into the MongoDB cloud database. For this, input parameters provide the given database connection string, the collection name, the device. Upon completion, the control returns to the GUI SIB, to display on the webpage the status of the workflow, and to be ready to accept further inputs.

There is no end SIB as this Web application does not terminate its execution: it is always available, as part of a device command and monitoring infrastructure.

### 5.2 Analytics Dashboard in R

For numerical and statistical analytics on the data, the Analytics Dashboard process shown in Fig. 9 implements a second web application. When the workflow starts, the SIB connection_mongoDB sends instructions to the R server along

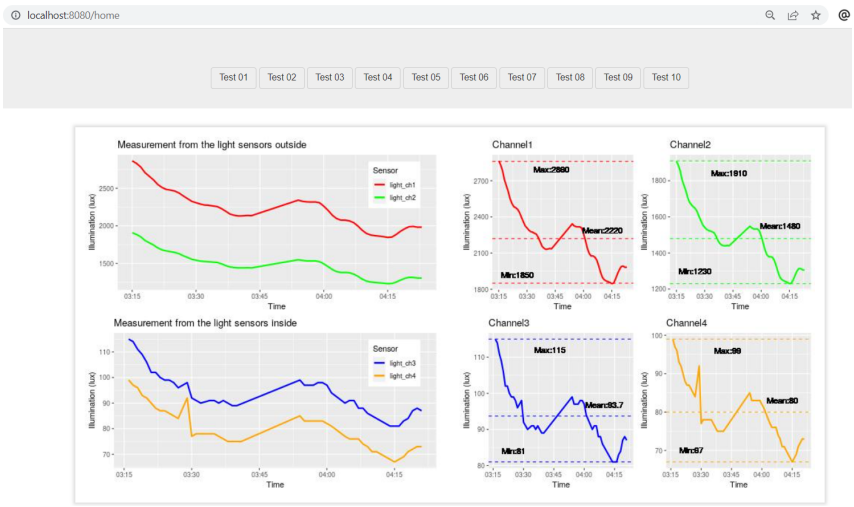



Fig. 8: IoT Web Application: the Data Visualization Dashboard

with the required inputs: connection string, database name and collection name of the MongoDB database. The R environment reads the dataset in JSON format using the *mongolite* package and returns the file handler for the dataframe. This file handler is passed to the split_data_frame SIB along with the column names that require splitting. The R server splits the master dataframe into multiple dataframes according to the given inputs. In our case they are light_ch1, light_ch2, light_ch3 and light_ch4, corresponding to the four channels in the two devices, two per each device. The file handler for the list of these four dataframes is passed to the next SIBs, plot_dual_linechart and plot_quad_summary_charts along with the variable names for the x and y axes. These SIBs use the four dataframes to generate the plots shown in Fig. 10 and Fig. 11. The plots are then shared with the analytics_dashboard GUI SIB that displays them on the web application dashboard, as shown in Fig. 8.

This is a simple pipeline in R that is easily extensible to more elaborate computations and visualizations. The advantage of having the data in the cloud is that it can be made accessible also to other analytics systems. We show how we carry out essentially the same computations in Python using another Low-code/no-code platform: Pyrus.

### 5.3 Analytics Dashboard in Python

Fig. 12 shows an alternative data analytics processing workflow pipeline that we implemented in the Pyrus platform, this time using Python as the language and platform of choice. The logic is similar to that in the DIME pipeline, but it is worth noting that Pyrus is itself a web application, and its pipeline modelling style is purely dataflow. In this sense, it is simpler to learn and has simpler

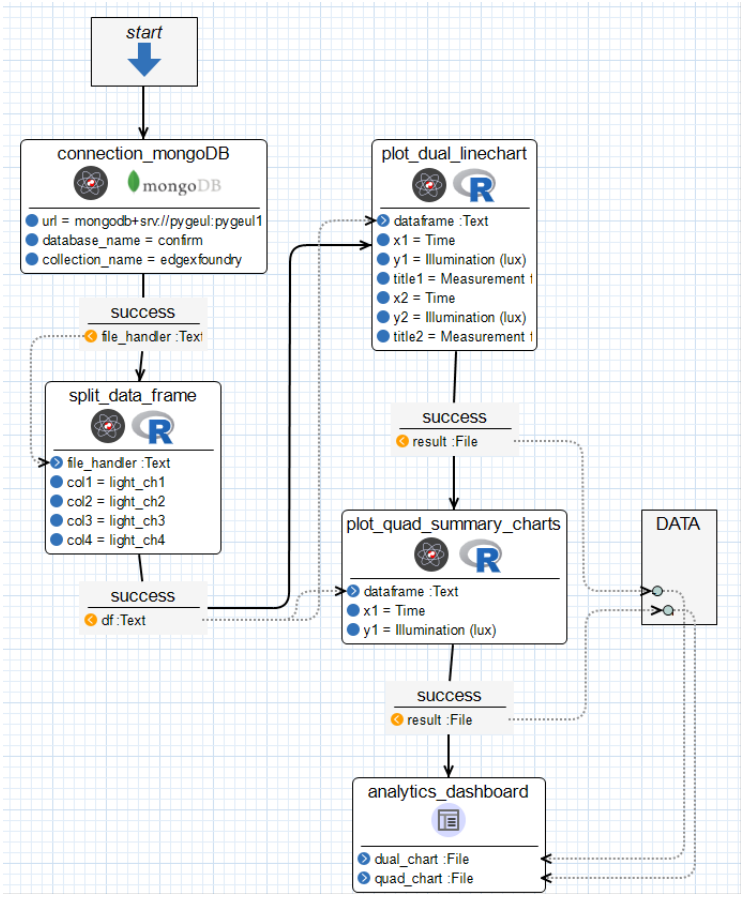



Fig. 9: Analytics Dashboard in R: the process in DIME

models, but it is also far less expressive and thus less powerful than DIME. Here, the Pyrus pipeline establishes a connection to the MongoDB database with the connection_to_mongoDB block, whose required inputs (i.e. MongoDB database URL, database name and collection) are provided as constant strings, which are the grey input blocks in the model. It fetches the requested data in JSON format and uses the Pandas package to create a dataframe. This dataframe is passed for preprocessing to the next block, convert_to_datetime: it converts the time of observation column to the correct time format for ease of analysis and for later plotting. The dataframe and the names of the channels are then passed to the block dataframe_split in order to create a separate dataframe for each light channel. Finally, the plot_all_data block plots them as graphs in the dashboard, as shown in Fig. 13. Here we see the same dual line plot and quad summary with minimum, maximum and mean value as in Figs. 10 and 11.

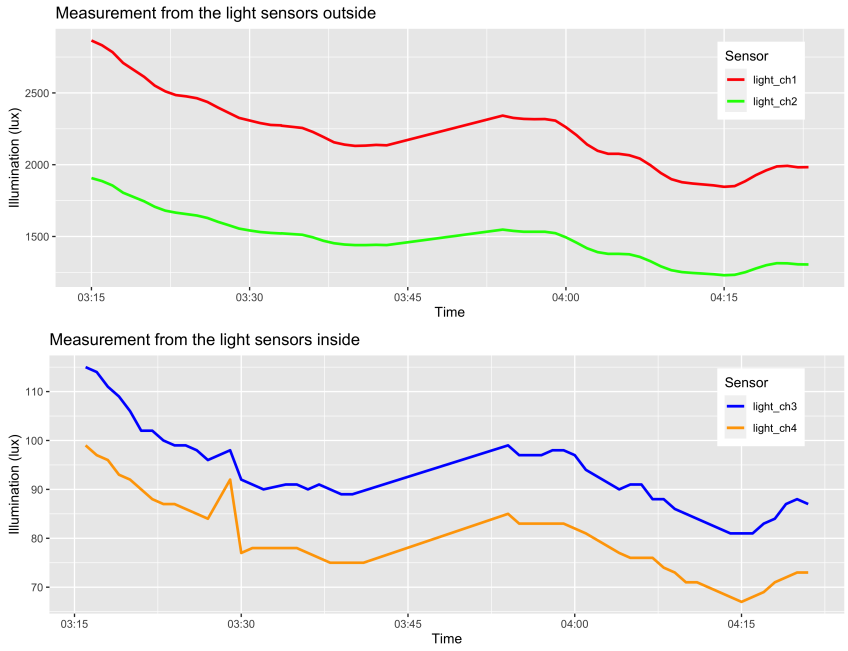



Fig. 10: Dual line plot - generated using R and the DIME DSLs and processes. Outside sensor (top) and inside sensor (bottom)

## 6 Results and Discussion

The case study described in this paper concretizes a heterogeneous architecture proposal for the development of low-code IoT applications involving cross-edge analytics. In this real-world use case, two devices interact with an orchestrator (here a simple Raspberry Pi) and an analytics server that includes a NoSQL database for storing the unstructured data. The purpose is to create several communication channels to send/receive data from the devices, and eventually provide a data analytics decision-making process that enables supervisors or an automated procedure to take decisions according to established criteria. In this particular case, we send instructions to different devices to start taking observations. We retrieve the observation data from the Pycom FiPy devices 1 and 2, and we show two alternative solutions for the analytics and graphical visualization: an R Server accessed through DIME processes, and a Python/Jupyter notebook server accessed through a Pyrus pipeline.

In the experimental setting, two Pycom FiPy devices installed in the Lero building at the University of Limerick record the observations at fixed intervals. FiPy-Pycom 1 collects light illumination measurements outside the building on channels `light_ch1` and `light_ch2`, while FiPy-Pycom 2 collects the respective light illumination measurements inside the room on `light_ch3` and `light_ch4`. This data is sent to a MQTT Broker that is part of the EdgeX Foundry frame-

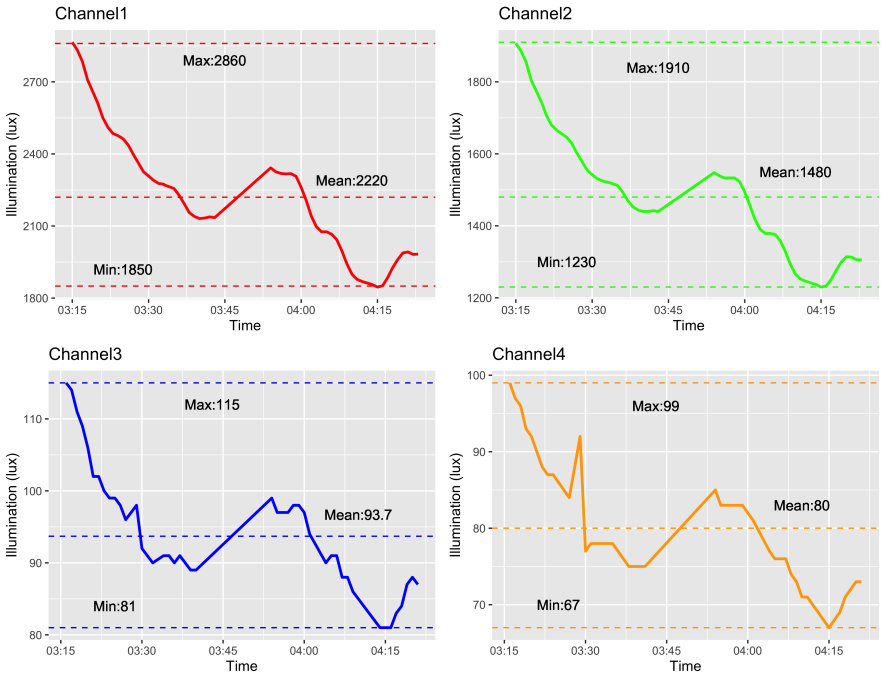



Fig. 11: Quad summary line plot - - generated using R and the DIME DSLs and processes. Individual channel plots, same colours as in Fig. 10

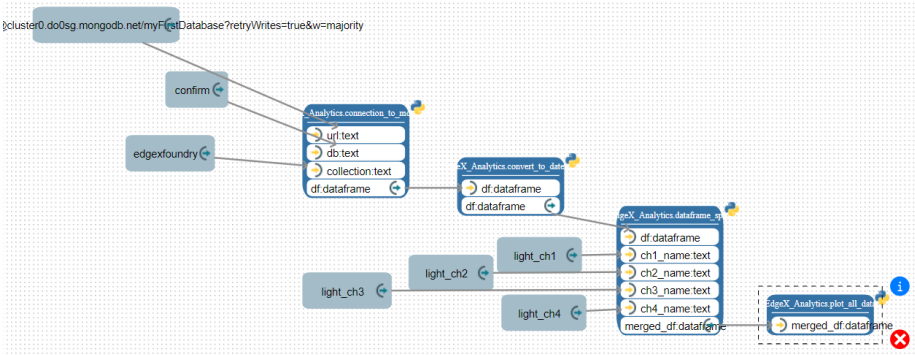



Fig. 12: Analytics Dashboard in Python: the Pyrus pipeline

work, running at the edge, in the orchestrator device which is the Raspberry Pi. When the data enters the EdgeX framework, we retrieve it through a REST API call and send it to an analytics pipeline where MongoDB (Fig. 6) stores the data, and then both an R and a Python servers provide the graphics interpretation, within a DIME process (Fig. 9) and a Pyrus process (Fig. 12, respectively.

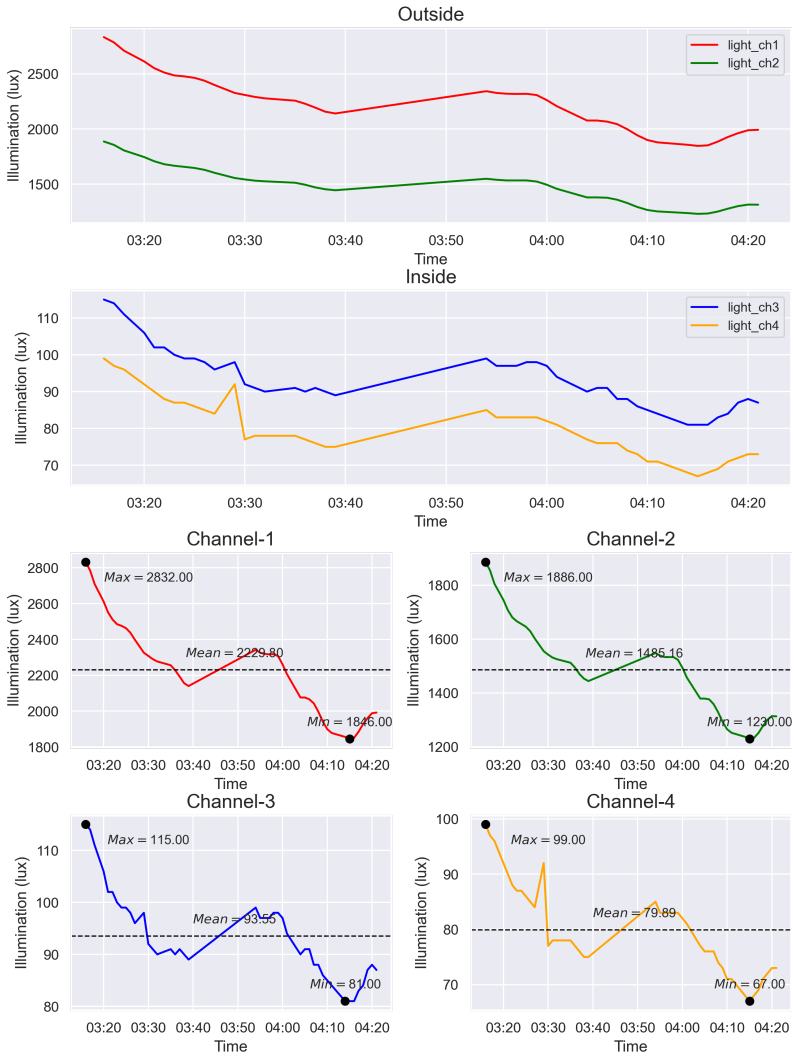



Fig. 13: Analytics visualizations - generated using Python and Pyrus

The presented use case shows the ease of modifying the requested features in a simple way: one only needs to modify the parameters of the SIBs, in a no-code way. This effectively converts the use case into an EAaaS (Edge Analytics as a Service), a service created by low-code/no code applications, which is a novelty in the research field.

The importance of this approach lies in its contribution to the Digital Thread Platform of Fig. 1: we have extended the Native DSLs for EdgeX Foundry,

REST, R, and MongoDB. We have also addressed MQTT and Raspberry Pi, but without the need to create specific native SIBs as they are covered by the EdgeX Foundry platform: from the DIME processes we simply communicate with the EdgeX Foundry services via REST, and we do not need to manually integrate them into DIME. As EdgeX Foundry runs on the Raspberry Pi within a docker container, the management is greatly simplified. For the light sensors, we use bespoke Python code that captures the light data from embedded digital light sensor and then sends this data to EdgeX foundry server using the MQTT protocol over WiFi .

The power of abstraction experienced through the use of EdgeX Foundry is again leveraged at the application design level through DIME: we create web applications without writing a single line of GUI code, we have as well reused several SIBs that were preexisting, for REST services and R, so that the combination of EdgeX Foundry and DIME appears as a powerful combination to provide a Digital Thread platform that supports low-code Internet of Things application development for Edge analytics. The quality and completeness of the base platforms are validated via empirical research, both within the open source community and with the adopters in academia [22, 11], industry [7], specifically also including manufacturing [34] and security [12]. The core approach for all these specific tools and applications is the language driven engineering (LDE) of [32]: it addresses the principles and the tools for balancing general purpose vs. purpose-specific design of languages and the generation of corresponding Integrated Modeling Environments, of which DIME and Pyrus are the ones we currently use most extensively. The more Native DSLs are included in DIME, the more diverse the systems it integrates, and the more applications are created, the more functionalities in each system, server, platform are made available for others to reuse.

In terms of relevance for the IoT user communities, this is a transformative contribution: instead of needing expertise in a large number of diverse technologies, the users needs to be only trained on these low code platforms. These tools leverage the opportunity to be more productive, efficient, and cost-effective delivering a solution, as the learning curve for abstractions is simpler than the one for programming languages. After initial training, IoT application developers can design, deploy, maintain and evolve their applications with less effort, in a uniform environment, at a more convenient abstraction level, and with better tool support for the code generation and deployment through platforms like DIME and Pyrus.

The goal of supporting non-experts in a specific domain or technology to achieve nevertheless good results in use cases that exceed their expertise is also important: with the proposed approaches, experts of some of the needed technologies create complex, heterogeneous applications using components produced and provided to the platforms by experts of other technologies and domains, widely extending their range of action and confidence.

## 7 Conclusion and Future Work

Domain specific languages in a low code, model driven paradigm have become a popular approach to design and develop heterogeneous systems. They empower non-software domain experts to participate in the development process. In this case study, we developed and deployed an IoT enabled edge analytics web application in two low code development environments. We used DIME for the application design and implementation of IoT and edge aspects as well as analytics in R, and Pyrus for data analytics in Python, demonstrating how such domain engineers can build innovative IoT applications without having full coding expertise. Our innovative platforms and development approach have the potential to simplify the development and deployment of such applications in industry.

Our next steps include the refinement of the baseline architecture and further extension of the supported devices, protocols and services. Particularly interesting are a DSL for orchestrating the EdgeX Foundry "cr" rule-based engine, which defines rules in a SQL-like syntax and creates streams to retrieve data from the Edge, as well as an extension to work with other orchestration engines like eKuiper from within DIME applications.

**Acknowledgements** This project received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Smart 4.0 Co-Fund, grant agreement No. 847577; and research grants from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (CONFIRM Centre), 13/RC/2094-1 (Lero, the Software Research Centre) and 18/CRT/6223 (SFI Centre of Research Training in AI).

## References

1. MongoDB Atlas Database | Multi-Cloud Database Service (Accessed, March 2022), https://www.mongodb.com/atlas/database
2. The Jupyter Notebook (Accessed, March 2022), https://jupyter.org
3. Ali, O., Ishak, M.K., Bhatti, M.K.L., Khan, I., Kim, K.I.: A comprehensive review of internet of things: Technology stack, middlewares, and fogedge computing interface. Sensors **22**(3) (2022)
4. AWS: Amazon web services iot (Accessed, March 2022), https://aws.amazon.com/iot/
5. Boßelmann, S., Frohme, M., Kopetzki, D., Lybecait, M., Naujokat, S., Neubauer, J., Wirkner, D., Zweihoff, P., Steffen, B.: Dime: A programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. pp. 809–832. LNCS 9953, Springer International Publishing, Cham (2016)
6. Chaudhary, H.A.A., Margaria, T.: Integration of micro-services as components in modeling environments for low code development. Proceedings of the Institute for System Programming of the RAS **33**(4) (2021)
7. Chaudhary, H.A.A., Margaria, T.: Dsl-based interoperability and integration in the smart manufacturing digital thread. Electronic Communications of the EASST **80** (2022)

8. Docker Inc.: Docker (Accessed, March 2022), https://www.docker.com/
9. EdgeX Foundry: The preferred edge IoT plug and play ecosystem - open source software platform (Accessed, March 2022), https://www.edgexfoundry.org/
10. Eoin Hinchy & Thomas Kinsella: Tines (Accessed, March 2022), http://www.tines.com
11. Gossen, F., Kühn, D., Margaria, T., Lamprecht, A.L.: Computational thinking: learning by doing with the cinco adventure game tool. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 990–999. IEEE (2018)
12. Gossen, F., Margaria, T., Neubauer, J., Steffen, B.: A model-driven and generative approach to holistic security. In: Resilience of Cyber-Physical Systems, pp. 123–147. Springer (2019)
13. H2o.ai: H2o.ai (Accessed, March 2022), https://www.h2o.ai/
14. Irmak, E., Bozdal, M.: Internet of things (iot): The most up-to-date challenges, architectures, emerging trends and potential opportunities. International Journal of Computer Applications **975**, 8887 (2017)
15. John, J., Ghosal, A., Margaria, T., Pesch, D.: Dsls and middleware platforms in a model-driven development approach for secure predictive maintenance systems in smart factories. In: ISoLA 2021. LNCS, vol. 13036, pp. 146–161 (2021)
16. John, J., Ghosal, A., Margaria, T., Pesch, D.: Dsls for model driven development of secure interoperable automation systems with edgex foundry. In: 2021 Forum on specification & Design Languages (FDL). pp. 1–8 (2021)
17. LITEON: Digital ambient light sensor (Accessed, March 2022), https://www.mouser.ie/datasheet/2/239/liteon_LTR-329ALS-01-1175539.pdf
18. Margaria, T., Chaudhary, H.A.A., Guevara, I., Ryan, S., Schieweck, A.: The interoperability challenge: building a model-driven digital thread platform for cps. In: ISoLA 2021. LNCS, vol. 13036, pp. 393–413. Springer (2021)
19. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. International Journal on Software Tools for Technology Transfer **5**(2), 107–123 (2004)
20. Margaria, T., Steffen, B.: Business process modeling in the jabc: the one-thing approach. In: Handbook of research on business process modeling, pp. 1–26. IGI Global (2009)
21. Margaria, T., Steffen, B.: Continuous model-driven engineering. Computer **42**(10), 106–109 (2009)
22. Margaria, T., Steffen, B.: extreme model-driven development (xmdd) technologies as a hands-on approach to software development without coding. Encyclopedia of Education and Information Technologies pp. 732–750 (2020)
23. Microsoft: Azure iot (Accessed, March 2022), https://www.microsoft.com/
24. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: Cinco: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. International Journal on Software Tools for Technology Transfer **20**, 1–28 (06 2018). https://doi.org/10.1007/s10009-017-0453-6
25. PTC: Thingworx (Accessed, March 2022), https://www.ptc.com/
26. Pycom: Fipy development modules (Accessed, March 2022), https://docs.pycom.io/datasheets/development/fipy/
27. Pycom: Pysense shield expansion module (Accessed, March 2022), https://docs.pycom.io/datasheets/expansionboards/pysense/
28. Siemens: Siemens mindsphere (Accessed, March 2022), https://siemens.mindsphere.io/

29. Simon Urbanek: Rserve - binary r server - rforge.net (Accessed, March 2022), https://www.rforge.net/Rserve/
30. Statista: Low-code development global platform market revenue (Accessed, March 2022), https://www.statista.com/statistics/1226179/low-code-development-platform-market-revenue-global/
31. Statista: Number of internet of things (iot) connected devices worldwide from 2019 to 2030 (Accessed, March 2022), https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/
32. Steffen, B., Gossen, F., Naujokat, S., Margaria, T.: Language-driven engineering: from general-purpose to purpose-specific languages. In: Computing and Software Science, pp. 311–344. Springer (2019)
33. The R Foundation: R: The r project for statistical computing (Accessed, March 2022), https://www.r-project.org/
34. Wortmann, N., Michel, M., Naujokat, S.: A fully model-based approach to software development for industrial centrifuges. In: International Symposium on Leveraging Applications of Formal Methods. pp. 774–783. Springer (2016)
35. Zweihoff, P., Steffen, B.: Pyrus: an online modeling environment for no-code data-analytics service composition. In: ISoLA 2021. LNCS, vol. 13036, pp. 18–40. Springer (2021)