A Model-Driven Approach for IoT-Based Monitoring Systems in Industry 4.0

Mahdi Ziaei[1] Bahman Zamani*[2] Ali Bohlooli[3]

[1]MDSE Research Group, Department of Software Engineering, University of Isfahan, Isfahan, Iran, Mahdi.ziaie@eng.ui.ac.ir
[2]MDSE Research Group, Department of Software Engineering, University of Isfahan, Isfahan, Iran, Zamani@eng.ui.ac.ir
[3]Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran, Bohlooli@eng.ui.ac.ir

Abstract— **Industry 4.0 provides a framework for applying new technologies in industrial environments to boost the efficiency and intelligence. A recently blossomed technology in Industry 4.0 is Internet of Things (IoT), which allows us to create a smart environment by connecting various equipment. One of the main applications of IoT in a smart factory is to design monitoring systems, which helps put the behavior of devices under permanent and comprehensive supervision. However, the rapid growth and change in the monitoring facilities creates a big challenge for people who either want to use that equipment in Industry 4.0, or want to update the systems to benefit from this technology. To address this problem, this paper presents new approach based on model-driven engineering paradigm, for simplifying the design and development of real-time monitoring systems in an industrial environment. Our approach includes a domain-specific modeling language, a graphical editor, and model-to-code transformations that generate a hardware descriptive code, a mobile application, and a web application for a monitoring system. To evaluate the applicability of our approach, a scenario in the power industry has been designed, which offers user a VHDL code, a mobile application, and a web application for monitoring processes of the plant.**

Keywords— Model-Driven Engineering, Industry 4.0, IoT, Monitoring Systems.

I. INTRODUCTION

One of the most important objectives in all industries, is a smart factory with high output efficiency. In recent years, variety of infrastructure has been introduced for implementing Industry 4.0 in various subindustries [1]. Industry 4.0 will create a framework where the most recent technologies, such as Cyber-Physical Systems (CPS), Internet of things (IoT), Internet of Services (IoS), and Internet of Data (IoD), can collaborate and interact to achieve the goals [2]. This collaboration leads to generate a smart yet flexible environment which is compatible with the industry changes [3]. Generally, Industry 4.0 is associated with the automation of electronic processes and Information Technology (IT) development. In the IT domain, the main focus of Industry 4.0 is on smart and communicating systems, including machine-to-machine and human-to-machine connection, as well as establishing required interactions for different data transmission in the distributed systems [4].

One of the approaches to achieve the Industry 4.0 goals is to apply IoT concepts in the industry. IoT, will bring a new atmosphere by establishing the connections, in other word, it will make all things around us alive [5]. IoT describes a world in which all things are connected and communicate smartly. This connection will be available through the internet or local network. Ultimately, this capability will make things to be intelligent within their environment. Accordingly, a network is generated which consists of several nodes associated with these things and the network will provide the ability to share data. Data shared on this network will be an initiation into new plans and scenarios like smart cities, health care supervision, smart transportation, and monitoring systems [6]. To provide capabilities and functionalities using IoT-based systems, particularly in monitoring systems, there is a need for service nodes, containers, subnodes such as sensors and controllers, and edge nodes (for communication) [7]. This connected service is generally developed by system experts. To develop an IoT-based system, one requires to have comprehensive knowledge of programming in both software and hardware [8].

Our main goal in this paper is to hide the complexities of developing IoT-based systems that are used as a monitoring system, from the users. It should be noted that, by the user, we mean the person who wants to design and develop a monitoring system in an industrial environment. To achieve the aforementioned goal and to facilitate the development process of such systems, we propose an approach based on the Model-Driven Engineering (MDE). MDE is a new approach in software engineering whose main concern is to reduce the software life cycle complexity by raising the abstraction level [5]. Our solution includes a Domain-Specific Modeling Language (DSML) for the domain of IoT-based industrial monitoring systems. The proposed language is based on a metamodel, which includes the general concepts in the domain of IoT-based monitoring systems, as well as the relationship between those concepts. To facilitate the modeling process for the user, a graphical editor has also been created. Finally, to generate the code automatically, some model-to-code transformations have been defined. Using these transformations, a model can be converted to the code of the monitoring system.

To build a monitoring system using our approach, the user first creates a model of the system using the proposed graphical editor. This model must conform to our metamodel. Then, the model created by the editor is given as input to the transformations, and as the output, several artifacts will be generated. Firstly, as the computational section, an executable Field-Programmable Gate Array (FPGA) code is generated. Secondly, as part of the cloud section, a web application is generated. Thirdly, an Android application is generated. The code generated for FPGA will be able to receive data from the equipment, send warning commands, and transfer information to the cloud service. The cloud section handles the high-speed data processing and web reporting. The Android application is

connected to the cloud section to send the user a failure message. The integration of the described collection of tools (metamodel, graphical editor, and code generator) will facilitate the development of real-time monitoring systems in the industry.

To evaluate the proposed approach, a scenario in the power generation industry has been considered. For this scenario, first of all, a model that describes the system requirements is created, and then the code is generated automatically from this model. The generated code demonstrates the applicability of the proposed approach.

The rest of this paper is organized as follows. Section II is a brief overview on the MDE approach. In Section III, we introduce a motivation example to show the problem, preferably as a real-world scenario. Section IV is dedicated to the related work. Section V presents our DSML for the modeling of industrial monitoring systems. In this section, we explain our metamodel, the constraints, the graphical editor, and the supported transformations. Section VI explains an example scenario that is used for the evaluation. Finally, we conclude the paper in Section VII.

II. Model-Driven Engineering

MDE is one of the methodologies available in the field of software design and development. In this method, the main artifact is the model and the whole process of the software life cycle is going on, around this main axis. The principle "everything is a model" expresses the importance of this concept in MDE [9]. In addition to the model, a new concept called metamodel is defined. Metamodels include general and abstract concepts of models [10]. To perform the modeling, a special modeling language must be designed. This modeling language is the metamodel and sometimes is called a DSML. As long as the language has an abstract level that is close to the problem domain, the complexity of the modeling process will be reduced and it is possible to produce better and easier to generate code automatically. Therefore, this approach allows its developers and users to be engaged with an abstraction level closer to their specific field of work. People who are novice in programming languages, or are often unfamiliar with those languages are enabled to generate their codes automatically. Using this approach, will ultimately reduce the complexity and improve system relationships. This attitude also has other significant benefits such as reducing product production time, which is the key to using generators code automatically and increasing the quality of the final product, as a result of using a DSML [11].

Another key element of MDE is model transformation. The importance of this pillar is such that it has been likened to the "transformation is heart of MDE" [12]. In the MDE software life cycle, the models will be converted to different levels of abstraction or code using transformation. In general, transformation is programs that receive one artifact as input and produce another as output. By considering the model and code as the products that can be produced in the model-driven methodology, in general, four categories of transformation can be introduced: model to model, model to code, code to model, and code to code [13].

III. Motivating Example

One of the most essential and important challenges for software designers and developers in the IoT field is about communicating and working with hardware [14]. To overcome this problem, the developers mostly use low-level programming languages to communicate with hardware devices, which in turn results in complex implementations [15].

To understand this challenge better, consider a scenario in the power industry and Combined Cycle Power Plants (CCPP). CCPP is a type of power plants that includes some gas turbines and steam turbines. In this type of power plant, they use the recovery generator (the heat from the exhausted gases of gas turbines, which can be up to 600 degrees Celsius) to produce vapor needed in the steam turbine production cycle. If the gas turbine is not a combined cycle, the exhaust gas enters the air directly. Nevertheless, the residue energy is wasted. Fortunately, in CCPP this energy is reused, and the steam turbine boiler (recovery generator) produces water vapor without using fuel. Therefore, it can be claimed that CCPP is a very efficient, flexible, reliable, cost-effective, and environment friendly solution for generating electricity.

By considering the importance of this operation in the power plant, it is necessary to create a monitoring system that works based on IoT. To do this, some sensors such as temperature, humidity, and vibration have been installed to receive data from the plant environment. There are warning facilities such as alarming system and danger lamp which are supposed to be activated in case of an emergency. All this equipment should be connected to a central controller for exchange, send, or receive of data. Data received from the plant ought to be saved in an external database to be retrieved once necessary on the cloud section. In this structure, it is also possible to send messages by a human operator using an Android application to report some errors that not recognition by embedded sensors as an input data for the cloud section. The processed data should be spontaneously delivered on a web dashboard platform to be put on display so that the user could be aware of his/her industrial environment on time.

The mentioned cluster of components will form an industrial monitoring system in a plant. However, implementing such systems has become difficult due to heterogeneous and various hardware device interactions. This diversity has disrupted the process of building new systems, because the system developer is supposed to have detailed and vast software and hardware knowledge. This has encouraged us to offer a new approach based on MDE for implementing real-time IoT based monitoring systems by considering the Industry 4.0 framework.

IV. Related Work

In recent years, several researchers have conducted in the field of IoT, CPS, IoS, and Industry 4.0. In this section, we will describe the researches that resemble significantly to our work and exist in the IoT domain and particularly IoT monitoring systems. In each part of this section, we will first give a brief description of the work done, and ultimately, will present the main distinction of our work.

Alulema et al. [6] provided a model-driven framework, including a DSML, a graphical editor, and model-to-code transformations, for the hardware nodes integration. This framework is designed and developed exclusively for IoT activities, such as smart homes. It uses special notations in this area, uses data extracted from environmental sensors, and stores it in external databases under the web protocol. The hardware platform in this research is Arduino and the language

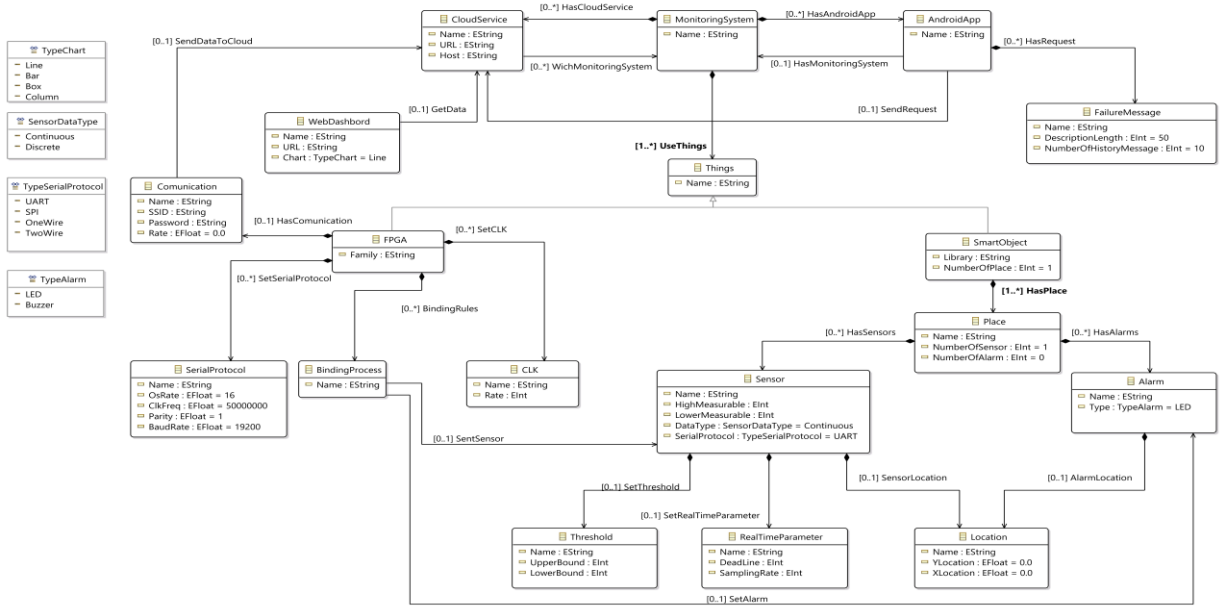


Fig. 1 The MetaModel Including both IoT and Industry 4.0 Concepts

of the generated code is C. But this research does not have the ability of real-time processing. This makes the work not beneficial in industrial environments which require such processing. In our work, by considering the FPGA hardware platform, cloud computing service, and paying special attention to the real-time parameters, we attempt to create a live monitoring system in an industrial environment. Such an environment often requires high-speed data processing.

Ciccozzi et al. [16] discussed the challenges of mission critical IoT systems. These challenges are divided into two categories: heterogeneity and associated complexity. To solve the mentioned challenges, the MDE has been used for the development process and runtime management of such systems. However, due to the broad domain and not considering sub-domains, the performance is deteriorated. However, in the present research, we have tried to make a proof-of-concept by limiting the domain of IoT to industrial monitoring environments. Limiting the domain has ultimately led to the increased accuracy in the modeling process and the achievement of more complete executable generated codes.

Molano et al. [5] introduced a comprehensive metamodel for communicating between the IoT, social networks, cloud, and Industry 4.0. In this research, special attention has been paid to the concept of industrial monitoring in IoT. However, the authors did not provide transformations for code generation. Thus, the future goal of their research is to generate automated code to perform regulatory monitoring processes in the Industry 4.0 and they aim to introduce Raspberry Pi board on hardware platform. In our work, in addition to providing a metamodel that covers all of the above concepts, it is possible for our users to generate the synthesizable final code on the FPGA hardware platform, web, and Android application using model-to-code transformations.

Einarsson et al. [7] introduced a DSML for modeling smart home applications. Their metamodel covered the concepts and architecture of a smart home. Also, by creating model-to-code language, it is achievable to create destination code from cross-platform to two instances of smart things and Alexa. The main differences between the current study and Einarsson et al.'s research can be considered as the domain of modeling Smart Homes vs. Industries, as well as a graphical editor (by our research) to simplify the modeling process and different target platforms.

V. The Proposed Approach

In this section, we will introduce a new approach for designing and developing an industrial monitoring system that relies on the Industry 4.0 framework. In our approach, the model-driven methodology is widely used to develop different parts of a monitoring system. This method consists of several components: a) metamodel, b) graphical editor, and c) mode-to-code transformations. In the following, we will present each part and explain how to use them in our approach.

A. MetaModel

In general, a metamodel is a model of models [10]. This means that a metamodel includes the definition of classes and their relationships. Fig.1 shows the metamodel provided for the domain of industrial monitoring systems. This metamodel consists of various components with different functionalities.

The **AndroidApp** class in Fig.1 contains information for developing the Android application needed in the monitoring system. This information includes number of messages in app history, to send errors that may not be detected by environmental sensors. The information in this category of errors must be injected into the system and be sent to the cloud section by a human operator using the application, and then be processed and notified to the user.

In the **CloudService** component, a service will be created that allows to store various information such as system state at different time, display a report to the user by a dashboard application, receive a failure message from the Android application, and make a connection with the FPGA board. In this service it is possible to receive data from sensors and perform some computational operations that require faster speeds and resources for processing and decision making in a monitoring system.

WebDashboard class gets the Uniform Resource Locator (URL) address from the user to set this address for the cloud section dashboard. Also, in this class, the type of diagram for

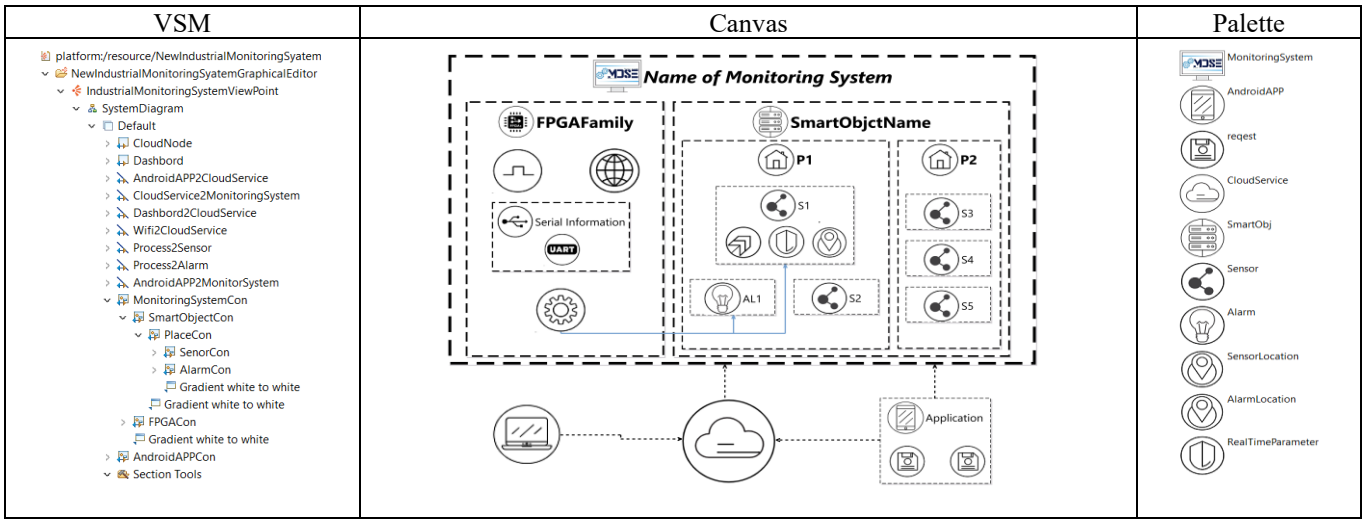


Fig. 2 Screenshot of Sirius for Developing the Graphical Editor.

displaying the sensor information to the user is determined. These types include Line chart, Bar chart, Box chart, and Column chart.

The monitoring system provided by this model includes some **Things**. According to IoT architecture, these objects are divided into two categories: smart object and computational object.

A **SmartObject** contains at least one sub-object. In each of them, there is one or more monitoring sensors or alarms. Also, each **Sensor** contains parameters to determine how it behaves. These parameters are distributed on the **Threshold** class, the **RealTimeParameter** class, and the **Location** class. In the Threshold class, we have upper and lower bound that represent the acceptable range for the sensor. In the RealTimeParameter class we have variables for real-time processing, such as a deadline and sampling rate, and in the Location class, there are parameters that determine the location of the sensor in the monitoring system. Note that, the location parameters are also used by the Alarm component.

In the computational section, we will have the **FPGA** component which is responsible for performing calculations, generating error commands, and sending information to the cloud section. The information required to communicate with the cloud is specified in the **Communication** meta class.

With our meta-model, it is possible to use several monitoring equipments, such as various types of sensors that work with different serial connections (SPI, UART, 1-Wire, and 2-Wire). The serial variables are defined with the help of **SerialProtocol** meta class. Another important function in this section is to establish a connection between the sensor and the alarm components. If a sensor exceeds the predefined threshold, the alarm must be activated. This is performed by using the **BindingProcess** component.

B. *Graphical Editor*

To simplify the task of system modeling for the users, a graphical editor is designed. This editor is implemented using the Sirius framework [17]. An overview of this editor can be seen in Fig. 2. By using Sirius, the elements that are defined as a class in the metamodel, can be drawn as graphical elements. It is also possible to edit the graphical elements and to create custom pallets using Sirius. Our editor is made up of three parts as follows:

a. Canvas, which is responsible for displaying and editing the information of the components designed in the editor.
b. Tool palette, which helps the user to add a component and some relationships to the project using the drag and drop operations.
c. View Specification Model (VSM), to configure classes and the relationships between them, as well as to create an icon for displaying each component.

C. *Model-to-Code Transformations*

To achieve the main goal of our research, which is generating code for an industrial monitoring system, we need to write a set of model-to-code transformations. We have used the Acceleo [18] language for writing the transformations. Acceleo is a template-based language that allows us to generate any kind of source code from Eclipse Modeling Framework (EMF) data source, automatically. This language is implemented on the metamodel. Our transformations, when executed, will receive a model that conforms to the metamodel and according to the created template code, the target model or code will be generated. We have used Acceleo to generate three types of code, automatically: Flutter, React JS, and VHDL. Flutter [19] is an open-source framework for mobile front-end design. Flutters contain some widgets. That each widget has some property, attributes, and children. React JS [20] is a javascript library for building web application user interfaces. In this language, by using front-end libraries, we will eventually reach a Single Page Application (SPA).

Very High-speed integrated circuit hardware Description Language (VHDL) [21] is one of the hardware description languages used in electronic and hardware design to describe digital systems such as FPGA. The structure of VHDL code for programing FPGA boards can generally be divided into two main parts:

a. Entity: in this section, the program port variables will be defined. These variables will be used in the next section.
b. Architecture: includes the main body of the program that will determine the behavior of the board according to various parameters.

Fig. 3 shows a piece of Acceleo transformation code for generating VHDL code, automatically. In this code, the URL

of the metamodel is written in line 2. Then, the information about the output file from the transformation operation is introduced (line 6). In the main body of the transformation code, the fixed and variable lines of the target code will be defined. Lines 7 to 9 include the introduction of the libraries required to execute the generated code on the specific platform. In lines 11 to 25, the part of Entity section in the VHDL language is introduced. In this section, some "for loops" are read to find the name of the instances in the Sensor, Alarm, and FPGA classes, and using these names, the requited input and output ports are defined.

```
1  [comment encoding = UTF-8 /]
2  [module Main('http://www.industrialMonitoringSyatem.com')]
3
4  [template public Main(aSmartObj : SmartObject)]
5  [comment @main/]
6  [file ('Main'.concat('.vhdl'), false, 'UTF-8')]
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.ALL;
9  USE ieee.std_logic_arith.ALL;
10
11 Entity MonitorSystem is
12     PORT(
13         [for (state : Sensor | Sensor.allInstances())]
14         [state.Name/]RX : IN STD_LOGIC;
15         [state.Name/]TX : OUT STD_LOGIC;
16         [/for]
17
18         [for (state1 : Alarm | Alarm.allInstances())]
19         [state1.Name/]Alarm : out std_logic;
20         [/for]
21
22         [for (state1 : FPGA | FPGA.allInstances())]
23         [state1.SetCLK.Name/] : out std_logic;
24         [/for]
25         S_LED_Default : out std_logic
```

Fig. 3 Partial Model-to-Code Transformation for VHDL

VI. Case Study

In the following, we will use the proposed approach and try to build a monitoring system for the case discussed in the motivation example section. A snapshot of the Android application and the web dashboard application which are automatically generated for the CCPP system, are illustrated in Fig. 4. In addition, the model of the CCPP system which is created using our graphical editor, as well as the corresponding VHDL code of this power plant which is generated automatically using our transformations, are shown in Fig. 5. This scenario involves: a) a FPGA board, which belongs to Spartan-6 family, to compute and control operations, b) cloud section for data processing, save monitoring information in external databases, and web reporting by dashboard, c) Android application to send failure messages, and d) five positions including gas generator, heat exchanger, combustion chamber, gas turbine, Waste Heat Recovery Units for Steam Generation (WHRU-SG), and steam turbine. In each position, according to the described parameters, the related sensors and alarms have been defined along with the parameters for determining their status. Multiple sensors in this scenario are not connected to any alarms. This indicates that the cloud system will use and show the data, sent by the sensors, in the dashboard section. However, the highly sensitive and risky situation of the sensors leads to the connection of the sensor and its corresponding alarm by the process node in FPGA section. In addition, FPGA section defines the required parameters for the communication and internal calculations, such as the serial communication protocol in the sensors, settings in the node for having interaction with devices, and the external clock adjustment to coordinate sending and receiving data. In the cloud section, service URL and host API are set. Regarding the Android app component, the structure of the messages related to the device failure and the corresponding relationships between the cloud and the monitoring system have been established.

The code generated for this case can be executed in the corrsponding platforms. For example, the generated VHDL code can be simulated in an appropriate software such as ISE Xilinx and can be synthesized in the destination platform.

In the process of generating FPGA, Flutter, and React JS required code, all the necessary aspects are considered to create the optimal code. Therefore, it can be claimed that the code generated in this approach will consume far less resources than other similar code. In fact, by taking advantage of the proposed approach, we can speed up the process of code generation and make it possible to access the code with high performing levels concerning their physical structures as well as their efficiency in various industrial environments.

Also, with the help of this method, people with low skill levels in the design and the development of monitoring systems can create an industrial monitoring system. This approach will accelerate the process of injecting Industry 4.0 into power plants and making smart plants by speeding up the code generation.

VII. Conclusions and Future Works

Today, the way industrial monitoring works is one of the major challenges in advanced industrial societies. Industrial monitoring has been smartened by IoT, especially by Industrial IoT (IIoT) and Industry 4.0. In recent years, facilities such as remote monitoring and control in factories are provided by the monitoring systems [22]. After analyzing some Industry 4.0 automation frameworks which are presented by other researchers, we concluded that there is a syntactical difference in most of these cases. However, all of them require the developers to be involved in the design and development stages. Considering that the information used in these systems significantly resemble each other, we recommend to proliferate from MDE to make the software life cycle in this area more advanced.

In the approach proposed in this paper, a DSML is defined for the industrial environment and a graphical editor is deployed. In addition, we will finally get access to generated code, automatically by running the model-to-code transformations that are provided as part of our approach. In this work, to generate the target code, we chose FPGA platform, Flutter, and React JS frameworks. This is because they offer some benefits such as: a) free and open-source code, b) expandable framework, c) modularity and component-based architecture, d) low resource consumption, e) strong documentation, f) large supporting community, and g) up-to date facilities. To evaluate the applicability of the proposed approach, we have modeled a monitoring scenario in the power industry (combined cycle power plant) and generated the code of a monitoring system, automatically.

As the future work we aim to a) expand this methodology and extend this architecture for generating code to be run on diverse hardware platforms such as Arduino and Raspberry Pi and b) enrich the automatic code generation for iOS mobile application.

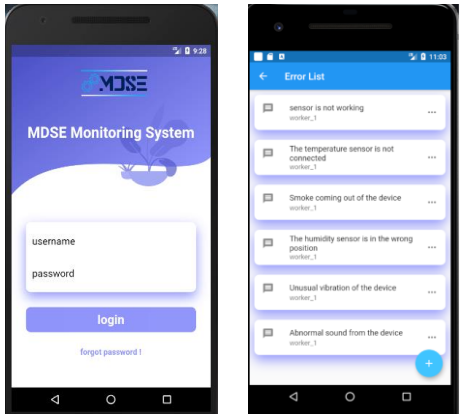
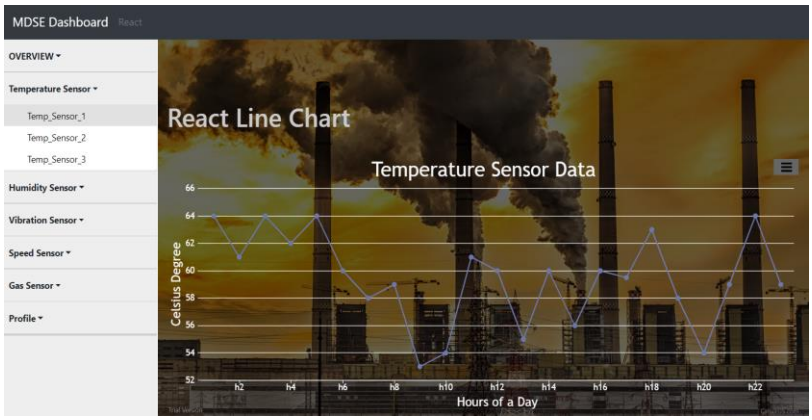
Mobile Application for Sending Message	Dashboard Page for Web Reporting from the Cloud
	

Fig. 4 Mobile Application and Web Dashboard Generated for CCPP

Generated VHDL Code for FPGA	CCPP System Architecture Model

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;

Entity MonitorSystem is
    PORT(
        STPressureSensor2RX   : IN STD_LOGIC;
        STPressureSensor2TX   : OUT STD_LOGIC;
        WGPressureSensorRX    : IN STD_LOGIC;
        WGPressureSensorTX    : OUT STD_LOGIC;
        CCTempratureSensor2RX : IN STD_LOGIC;
        CCTempratureSensor2TX : OUT STD_LOGIC;
        GTPressureSensor2RX   : IN STD_LOGIC;
        GTPressureSensor2TX   : OUT STD_LOGIC;
        GTTempratureSensor1RX : IN STD_LOGIC;
        GTTempratureSensor1TX : OUT STD_LOGIC;
        CCPressureSensorRX    : IN STD_LOGIC;
        CCPressureSensorTX    : OUT STD_LOGIC;
        GTPressureSensor1RX   : IN STD_LOGIC;
        GTPressureSensor1TX   : OUT STD_LOGIC;
        STPressureSensor1RX   : IN STD_LOGIC;
        STPressureSensor1TX   : OUT STD_LOGIC;
        HETempratureSensorRX  : IN STD_LOGIC;
        HETempratureSensorTX  : OUT STD_LOGIC;
        GGTempratureSensorRX  : IN STD_LOGIC;
        GGTempratureSensorTX  : OUT STD_LOGIC;
        STHumiditySensorRX    : IN STD_LOGIC;
        STHumiditySensorTX    : OUT STD_LOGIC;
        CCTempratureSensor1RX : IN STD_LOGIC;
        CCTempratureSensor1TX : OUT STD_LOGIC;
        WGTempratureSensor2RX : IN STD_LOGIC;
        WGTempratureSensor2TX : OUT STD_LOGIC;
        STTempratureSensor2RX : IN STD_LOGIC;
        STTempratureSensor2TX : OUT STD_LOGIC;
        GGPressureSensorRX    : IN STD_LOGIC;
        GGPressureSensorTX    : OUT STD_LOGIC;
        HETempratureSensor2RX : IN STD_LOGIC;
        HETempratureSensor2TX : OUT STD_LOGIC;
        STTempratureSensor1RX : IN STD_LOGIC;
        STTempratureSensor1TX : OUT STD_LOGIC;
        GTTempratureSensor2RX : IN STD_LOGIC;
        GTTempratureSensor2TX : OUT STD_LOGIC;
        WGTempratureSensor1RX : IN STD_LOGIC;
        WGTempratureSensor1TX : OUT STD_LOGIC;
        GTSpeedSensorRX       : IN STD_LOGIC;
        GTSpeedSensorTX       : OUT STD_LOGIC;
        GTTemp1AlarmAlarm     : out std_logic;
        GTPressure1AlarmAlarm : out std_logic;
```


Fig. 5 CCPP System Architecture and the Generated VHDL Code for the FPGA Board

REFERENCES

[1] M. Hermann , T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, 2016.

[2] K. Suri, A. Cuccuru, J. Cadavid, S. Gerard, W. Gaaloul, and S. Tata, "Model-based development of modular complex systems for accomplishing system integration for industry 4.0," in *5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, Porto, 2017.

[3] L. D. Xu, E. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research,* vol. 56, no. 8, pp. 2941-2962, 2018.

[4] v. alcacer and v. Cruz-Machado, "Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems," *Engineering Science and Technology, an International Journal,* vol. 22, no. 3, pp. 899-919, 2019.

[5] J. I. R. Molano, J. M. C. Lovelle, C. E. Montenegro, J. R. Granados, and R. G. Crespo, "Metamodel for integration of Internet of Things, Social Networks, the Cloud and Industry 4.0," *Journal of Ambient Intelligence and Humanized Computing,* vol. 9, no. 3, p. 709–723, 2018.

[6] D. Alulema, J. Criado, and L. Iribarne, "A Model-Driven Approach for the Integration of Hardware Nodes in the IoT," in *World Conference on Information Systems and Technologies*, Galicia, 2019.

[7] A. Einarsson , P. Patreksson, M. Hamdaqa, and A. Hamou-Lhadj, "SmartHomeML: Towards a Domain-Specific Modeling Language for Creating Smart Home Applications," in *2017 IEEE International Congress on Internet of Things (ICIOT)*, Honolulu, 2017.

[8] C. M. Sosa-Reyna, E. Tello-Leal, and D. Lara, "Methodology for the model-driven development of service oriented IoT applications," *Journal of Systems Architecture,* vol. 90, pp. 15-22, 2018.

[9] J. Bezivin, "In search of a basic principle for Model Driven Engineering," UPGRADE – *The European Journal for the Informatics Professional*, vol. 5, no. 2, pp. 21-24, 2004.

[10] L. Samimi-Dehkordi, B. Zamani, and S. Kolahdouz-Rahimi, "Bidirectional Model Transformation Approaches," in *6th International Conference on Computer and Knowledge Engineering (ICCKE 2016)*, mashhad, 2016.

[11] J. Luoma, S. Kelly, and J.-P. Tolvanen, "Defining Domain-Specific Modeling Languages: Collected Experiences," in *4th workshop on Domain-Specific Modeling*, 2004.

[12] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," in *IEEE Software*, vol. 20, no. 5, pp. 42-45, Sept.-Oct. 2003, doi: 10.1109/MS.2003.1231150.

[13] B. Zamani, On Verifying the Use of a Pattern Language in Model Driven Design, PhD diss., Concordia University, 2009.

[14] T. Gomes , P. Lopes, J. Alves, P. Mestre, J. Cabral, J. Monteiro, and A. Tavares, "A modeling domain-specific language for IoT-enabled operating systems," in *43rd Annual Conference of the IEEE Industrial Electronics Society*, Beijing, 2017.

[15] S. Teixeira, B. A. Agrizzi, J. G. Filho, S. Rossetto, and R. d. L. Baldam, "Modeling and automatic code generation for wireless sensor network applications using model-driven or business process approaches: A systematic mapping study," *Journal of Systems and Software,* vol. 132, pp. 50-71, 2017.

[16] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, and R. Spalazzese, "Model-Driven Engineering for Mission-Critical IoT Systems," *IEEE Software,* vol. 34, no. 1, pp. 46-53, 2017.

[17] Sirius Official Web Site, "The easiest way to get your own Modeling Tool," The Eclipse Foundation, [Online]. Available: https://www.eclipse.org/sirius/. [Accessed 10 May 2020].

[18] Acceleo Official Web Site , "EASILY CREATE CUSTOM CODE GENERATORS, "Eclipse Foundation, [Online]. Available: https://www.eclipse.org/acceleo/. [Accessed 10 May 2020].

[19] Flutter Official Web Site , " Design beautiful apps, " Google's UI, [Online]. Available: https://www. https://flutter.dev/. [Accessed 10 May 2020].

[20] React Official Web Site , " A JavaScript library for building user interfaces, " Facebook Inc, [Online]. Available: https://www. https:// https://reactjs.org//. [Accessed 10 May 2020].

[21] C. Z. Myint, L. Gopal, and Y. L. Aung, "Reconfigurable smart water quality monitoring system in IoT environment," *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, Wuhan, 2017.

[22] S. Aheleroff, X. Xu, Y. Lu, M. Aristizabal, J. P. Velásquez, B. Joa, and Y. Valencia, "IoT-enabled smart appliances under industry 4.0: A case study," *Advanced Engineering Informatics,* vol. 43, 2020.