Applying MDE for Healthcare Systems

Meriem Ouederni[1], Fan Qiyue[1], Quanrui Mu[1] and Lotfi Chaari[1]

University of Toulouse, INP, IRIT, France
firstname.lastname@toulouse-inp.fr

Abstract. In this work, we apply Model-Driven Engineering to propose a new Domain Specific Language for IoT-based healthcare systems. We particularly focus on data synchronisation between EEG and ECG devices. At the very first step, we suggest a meta-model for IoT framework and then we refine it to get the one adapted for healthcare systems. Farther, our proposal enables to automatically generate java implementation through the application of Model-to-Text (M2T) transformation. Thus, we get up with the executable code witch enables us to validate real world case studies. Furthermore, our approach allows us to perform in the future another transformations into formal languages such as LOTOS (algebraic languages), and this is in order to formally check some required properties.

Keywords : IoT, Healthcare, Distributed System, Model Driven Engineering, Synchronization

1 Introduction

Doctors today are spending less time with patients due to their ever increasing number, and this happens with more often inefficient medical devices. Such a situation does not help to save human-life as best as we hope, in particular in several cases where the medical system does not provide the doctor with the accurate data at the good moment. Hence, good decision could be missed several times.
Moreover, continuous and real time home surveillance of vulnerable or isolated patient is difficult using traditional tools and medical devices. In this sense, using Internet of Things (IoT) devices helps to reach such objectives. Regarding IoT and their application in healthcare systems, there are usually several interconnected devices which are sending separately data (modelled as signals) to monitors. Monitors are used to collect and analyse patient data and make decision. However, today's technology is missing synchronisation of all received data. This feature if done could be crucial to reduce the amount of time, yet also possible frustration and confusion, when taking care of patients.
In multi-system signal transmission, the synchronization has always been a problem that needs to be solved, especially for IoT devices. In order to obtain data describing a state in a specific and accurate time, the signal receiver (*i.e.* monitor) must synchronize the signals from different devices depending on the clock,

otherwise the output signal has a phase gap on the clock, which will cause incorrect judgments when multiple data are implied.
Model-Driven Engineering (MDE) bridges the gap between problems and solutions. It enables early validation and verification of implied systems. There are already relevant works about IoT system modeling. Pramudianto et al. [11] presented an architectural prototype for IoT development which separates the domain modeling from technological implementations. However, the data has not been modeled and then synchronisation has not been considered. In [9], the authors proposed a framework for IoT applications which allows user to model IoT systems with a JSON file while the meta-model they used was not generic enough.
The main contributions of our paper are as follows :

- We bring together several keys from Model-based Software Engineering and IoT modelling in order to assist healthcare professionals and improve patient assistance. We apply our approach to IoT-based healthcare systems and we resolve the signal synchronization problem.
- We provide the domain experts (*i.e.* doctors) with a Domain-Specific Language (DSL) where the front-end language enables experts to detect critical health issues from the collected and analysed data.
- We reduce the gap between both problems and solutions of data synchronisation.

In order to illustrate our approach, we consider a case study including health monitoring using Electrocardiography (ECG) and Electroencephalography (EEG) signals. Specifically, we address the signal synchronisation issue when jointly handling these data. In this case, synchronization is crucial mainly because the two signals are generally recorded at different temporal resolutions.
The rest of this paper is organized as follows. In Section 2, the background of IoT, synchronization and MDE is given. The proposed language, called here iDSL, is developed in Section 3. The model-to-text approach is then detailed in Section 4, while conclusions are provided in Section 5.

2 Background

2.1 Internet of Things

The IoT can be defined as a collection of physical objects like softwares, sensors or devices, whose network connectivity is ensured to allow these objects to interact, collect and exchange data[5, 14]. The use of IoT is nowadays guided by increasing economical, industrial and societal needs, especially in the fields of safety [7], industrial control [16] and human health[8].
Specifically, using IoT for healthcare systems is being widely considered, and this for different applications, especially in personalized healthcare [1].

In the field of Model-Driven Engineering, there are already some platforms and applications developed for IoT :

- MARTE (Modeling and Analysis of Real-Time and Embedded systems)[3] provides support for specification, design, and verification/validation stages. These are then refined for both modeling and analyzing concerns. Modeling parts provide support required from specification to detailed design of real-time and embedded characteristics of systems. Analyzing parts provide facilities to annotate models with informations required to perform specific analysis, especially, on performance and schedulability.
- openHAB[13] represents a general-purpose framework for smart home gateways and is fully based on the Eclipse SmartHome project, which allows building smart home solutions as the most common application area of IoT. The heterogeneity and connectivity are supported by OSGi plug-ins called bindings and APIs in the OpenHAB framework.
- ThingML (Internet of Things Modeling Language)[6] is an MDE approach that aims to cover a considerable amount of IoT use cases. The main contribution of the project is a code generation framework and an associated methodology to give practitioners full control of the code by letting them easily customize compilers for their needs. To accomplish this, expertise in MDE and Domain Specific Language (DSL) is required.

2.2 Signal Synchronisation

When using multiple IoT devices for system monitoring, two main issues arise. The first one is related to the analysis of collected data. The second one is related to the synchronization between different devices serving to collect data. Indeed, if the state of the monitored system is analyzed through multiple sensors, one needs to be sure that data collected at a time stamp t using the different devices reflect the state of the state at this same time stamp. Generally speaking, some delay may occur due to acquisition and transmission environments. Synchronizing all collected data is therefore essential for the sake of monitoring efficiency.
Such synchronization is mainly important for healthcare systems. In his paper, we focus on health monitoring using connected electroencephalography (EEG) and electrocardiography (ECG) devices [4]. A simultaneous data collection is performed using these two sensors in order to asses the patient state in real time. In [2, 4], the collected data is pre-processed to extract signal windows corresponding to a given duration of recording. Ensuring that the recording performed using the different sensors corresponds exactly to the subject state at the precise timestamp is therefore necessary.

2.3 Model Driven Engineering

MDE (Dodel Driven Engineering) aims at developing complex systems guided by models which rely on high abstraction level. This is achieved based on a number of principles that involve the concepts of model (M1), metamodel (M2), metametamodel (M2) and model transformations. Fig. 1 illustrates those different models *w.r.t* OMG (Object Modelling Group) classification.

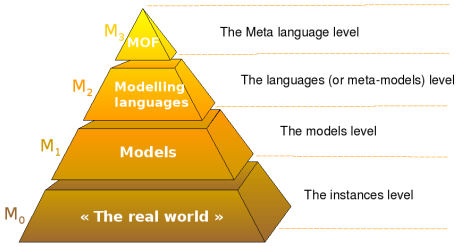

Fig. 1. OMG Pyramid.

MDE produces well structured and maintainable systems and increases the level of abstraction. Such a technique consists in the following.

- It enables to inter-operate with and execute models.
- Using EMOF (Eclipse Modeling Framework)[1], models can be exchanged throughout their serializable version encoded into XMI (Xml Metadata Interchange).
- It uses source code to automatically generate target code implementing huge software systems and this alleviates their complexity. This can be performed using well known Model-to-Text (M2T) and/or Model-to-Model (M2M) transformations.
- It also ensures early validation, *e.g.* using OCL (Object Constraint Language) [10], and thus we avoid costly systems bugs.

2.4 Case Study

In order to illustrate the proposed solution for synchronization, case study involving three devices is considered: EEG sensor, ECG sensor and a monitor host computer. Let us then consider the following scenario.

1) The monitor first receives data from the ECG device, it records the time when the first ECG data arrived and calls it "monitor-clock-ecg". We then get the first ECG data's local time-stamp, call it "clock-ecg".
2) After a few moment, the monitor receives data from the EEG device during one second. The monitor records the time when the first EEG data arrived, calls it "monitor-clock-eeg", and gets the first EEG data's local time-stamp: "clock-eeg".
3) The ECG and EEG devices may have different sampling rates. The monitor can get them from both devices. We assume that the sampling rates of the two devices are both accurate without drift.
4) We also assume that while offsets for ECG and EEG are different from each other, they remain constant during the acquisition/synchronization process.
5) While keeping running, the monitor performs the synchronization process once starting to receive data from both devices.

[1] `https://www.eclipse.org/modeling/emf/`

3 The iDSL design

3.1 Meta-modeling

Our aim here is to propose a meta-model (M2 level according to Fig. 1) as generic as possible. By doing so, we guarantee the separating domain modeling from concrete implementations. The proposed meta-model is displayed in Fig. 2.

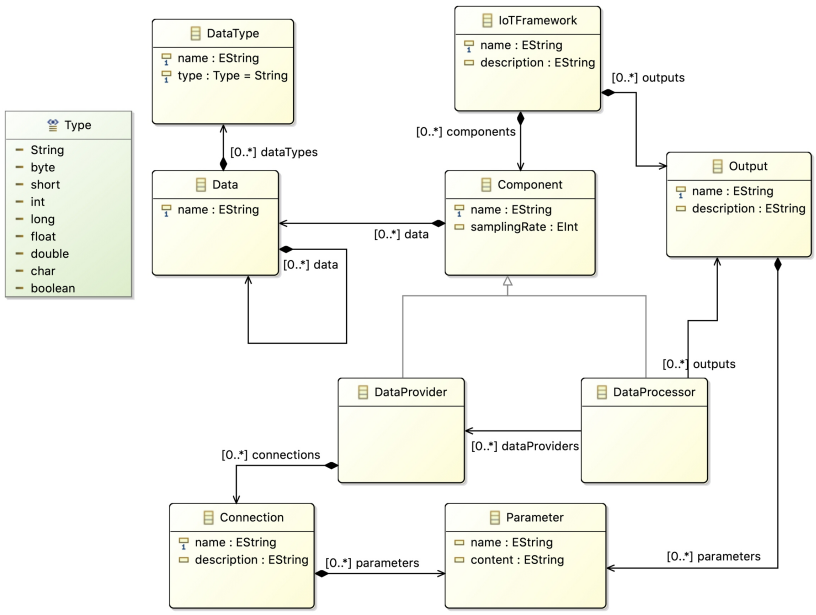

Fig. 2. iDSL Meta-Model.

In Fig. 2, the following explanations are provided to help understanding the proposer meta-model.

- **IoTFramework** : The entry of a model. It contains a name and a description. Other elements can only be created under one IoTFramework.
- **Component** : Common characteristics between data providers and data processors are summarized and put in the Component class, which includes name, sampling rate and data structure.
- **DataProvider**: We model different data collecting devices as DataProvider. In our study case, both ECG and EEG devices are DataProvider. They have name, sampling rate, connection type and data structure. Connection type and data structure can be specified in their own class.
- **Connection**: As we know, IoT systems are known for their heterogeneity. One of which is the diversity of connections. We model the connections by

allowing users to add parameters for their own connection. The proposed model supports the fact that one device may have multiple types of connection.
- **Data**: With Data class, users can create their own data structure for a specific device. This data structure consist of a series of atomic data type.
- **DataProcessor**: The component where the signal processing is performed. In our framework, concrete processings need to be implemented.
- **Output**: Systems may have different types of outputs. Here, the output can consist of multiple parameters, each parameter has its name and content. In the implementation part, one can have enough information about how to execute.

It is worth noting that the proposed meta-model is still passive and not operational. This means that it only describes the structure of our destination DSL. At this level, it does not enable to check static information, *e.g.* constraints to be satisfied (see Sec. 3.4).

3.2 Model

According to our meta-model and the actual requirements applied to ECG and EEG signal synchronization, we can instantiate a specific model whose UML class diagram is shown on Fig. 3. We make sure that it is a model (M1 level) which is conform *w.r.t* the meta-model of Fig. 2.

In Fig. 3, the following explanations are provided to help understanding the proposer model.

- **EcgSensor/EegSensor**: instantiated by DataProvider, they are used as thread class to establish connection with ECG and EEG devices, which allows us to obtain data for the following process.
- **WiFiConnection/BluetoothConnection**: instantiated by Connection, they provide a series of parameters corresponding to a device. Indeed, for different methods of connection like TCP or UDP, the mandatory parameters are different. The parameter attributes are instantiated by Parameter.
- **EcgSignalWindow/EegSignalWindow**: instantiated by Data, they are data entities that encapsulate a series of attributes of the signal information provided by the device. These attributes are necessary to describe the recorded signals. As different devices provide different signal formats or structures, the attributes inside a data entity are instantiated by DataType.
- **EcgSignal/EegSignal**: instantiated by Data, they are data entities that encapsulate the signal value with timestamp. The signal value with timestamp is one of those integral parts of the signal window which also includes other informations about the signal format, and consequently this makes use of the self-composition relationship of Data in meta-model.
- **Synchronizer**: instantiated by DataProcessor, it is the the most important part of the this model which aims to establish a framework for synchronizing ECG and EEG signals. Due to the methods provided by this class, we can

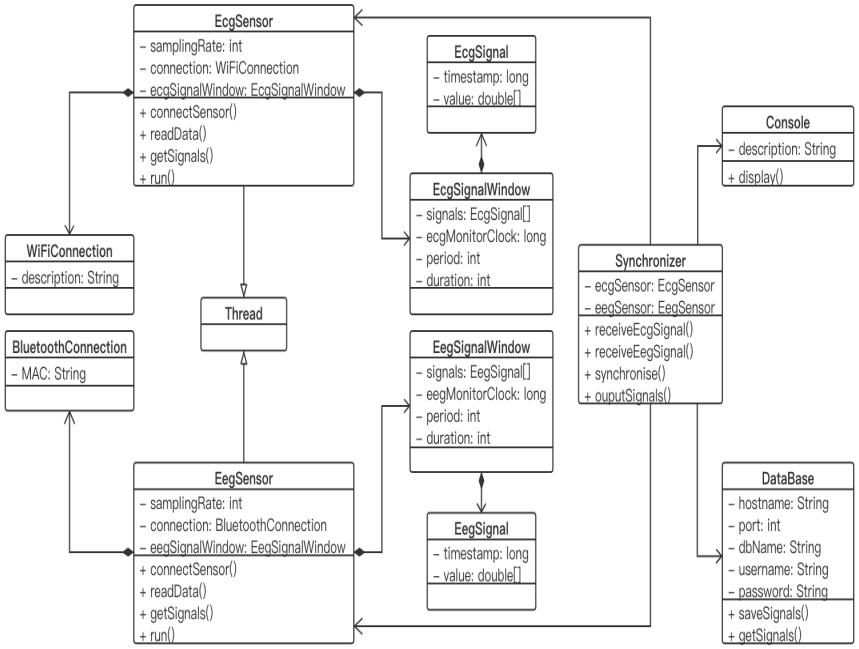

Fig. 3. A Model for iDSL.

find the time when the signals of both devices are obtained for the first time with corresponding local clocks and then output following signals according to the difference between local clock and device clock of each device. This enables us to obtain signals from both devices at the same absolute time after the process of synchronization.

– **Console/DataBase**: instantiated by Output, they are responsible for output or display of the signals after synchronization. Similar to Connection, different methods of output require different attributes of parameters. Those attributes in those classes are instantiated by Parameter.

3.3 Concrete Syntax

In this Section, we develop the adopted textual and graphical syntax.

Textual Syntax In order to let domain experts get started faster, our iDSL is inspired by the XML format. A tag is a markup construct that begins with "<" and ends with '>', which indicates a class or an attribute in the Meta-model. An element is a logical document component that begins with a start-tag and ends with a matching end-tag. The characters between the start-tag and end-tag are the element's content, and may contain markup, including other elements, which

are called child elements. Tag and content meet the rules of key-value pairs. An example is provided below.

```
<Connection>
    bluetooth-ecg
    <description>'ecg-bluetooth-connection'</description>
    <parameter> MAC <content>'AA:12:34:AA:12:34'> </content></parameter>
</Connection>
```

Listing 1: Tag example using the proposed textual syntax.

Domain experts can therefore easily do their modeling step.

Graphical Syntax An example of graphical model related to the ECG/EEG case study, specifically signal synchronization that conforms to the meta-model with Sirius tool[15], is illustrated in Fig. 4.

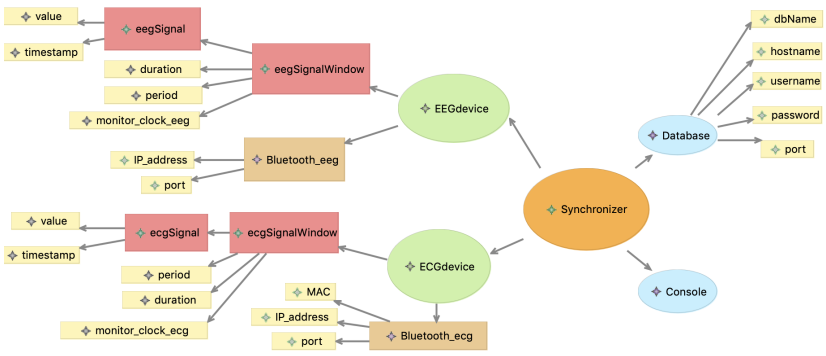

Fig. 4. An Example of graphical Syntax of ECG/EEG synchronization case study.

Hereafter, the main features of our graphical syntax are commented.

- ECG and EEG devices are two data providers according to the meta-model, both of them have specific data structure and connection types.
- ECG/EEG SignalWindow corresponds to the data structure of the two signals respectively. Both have attributes of duration and period and sub structure of ECG/EEG Signal.
- ECG/EEG Signal represents the signal structure collected and transmitted by IoT devices. It should be pointed out that although both of them contain time-stamps and values, they are different in data types.

– Synchroniser is where we preprocess the signals, that is, the synchronization operation. It receives signals from both devices, performs synchronization, and outputs as required.

3.4 OCL-based Validation

Object Constraint Language is used to create constraints on each class in the meta-model design of MDE. It is obvious that descriptions of complex and comprehensive constraints cannot be defined directly when creating a meta-model. OCL is essential to improve the model standardization instantiated from our meta-model. Therefore, the following OCL constraints have been established.

– Name specification of IoTFramework, Component and Output: the name must not be empty, and it should be greater than two characters, start with a letter and consist of both letters and numbers.
– DataProvider: the value of samplingRate of DataProvider must be positive.
– DataProcessor: if DataProcessor has at least one related Output, it must have at least one related DataProvider. SamplingRate of DataProcessor must be between the maximum value and minimum value of all the related DataProviders. All the related DataProviders of the DataProcessor must have at least one Connection and at least one Data. Related Data of DataProcessor must include those Data of its related DataProviders.
– Data: Data must have at least one DataType or at least one Data with DataType. Data should not have two identical DataTypes. The Data can not have two-level nesting.
– Connection and Output: each Connection or Output must not have two Parameters with the same name.
– Parameter: if the name of the parameter is MAC or IP, its content must conform to the correct format (defined by regular expression).

Using an OCL, we successfully defined the above-mentioned constraints with detailed descriptions. Each time a model is instantiated from ourmeta-model, the validation of OCL constraints is a prerequisite to make sure that the model meets our standards. If not, the corresponding error and its location will be prompted.

4 Model-to-Text Transformation

After the meta-model and model definition, model-to-text transformation aims at automatically generating executable code and rigorously checking constraints. We therefore need to ensure the transformation from the model into both Java and formal language LOTOS, respectively. To do so, we use the Acceleo utility from Obeo[2]. Through these transformations, experts in different domains can perform different operations on models that are conform to the meta-model from their own research perspectives.

[2] https://www.acceleo.org, last accessed 10 Sept. 2021

4.1 iDSL2Java

We first focus on the transformation from model to Java code. According to the meta-model, we use the Acceleo tool to write conversion code and realize the transformation from model to text (M2T), and automatically generate the whole framework of a project model, including all classes and attributes as well as getters/setters for all attributes. What needs to be pointed out is, from the model, one can only get the main framework of the Java project, including the definition of classes and attributes, however, methods need to be added later.

4.2 Signal Synchronization

After getting the automatically generated framework, synchronization involves three main steps. The first one is to connect to the IoT devices, i.e. both ECG and EEG. The second step is to extract signals' time-stamp and perform synchronization operation. Finally, the third step is to output the synchronized signals according to the model.

Connect to devices
One of the main heterogeneities in IoT is the diversity of device connection methods. Popular methods include Bluetooth, ZigBee, WiFi, Lora, NB-IoT, etc. In our simulated scenario, we assume that the connection has been established and we use Socket with C/S mode to exchange data. The monitor acts as server to keep listening the connection request from clients. Both ECG and EEG devices act as client. Once connected to the monitor, they will send out signals which contain local clock (time-stamps) and data collected from sensors.
Since IoT devices may have different local clock, our synchronization approach is based on recording the current monitor local time when the monitor receives a signal's first data, and using it as a reference to calibrate the local clock of the IoT devices.

Synchronization
In our study case, ECG and EEG signals are different in nature, meaning that the EEG signal is continuous while ECG is event-driven, and therefore discontinuous. In other words, the monitor performs the synchronization only when it receives signals from both devices at the same time.
To do so, we need parameters include local time clock, sampling rate and monitor recorded clock of both continuous and discontinuous signals. We define an offset as a monitor recorded clock minus a local time clock. What is needed is the local time-stamp of the continuous signal that actually corresponds to the discontinuous signal's local clock, call as "head". It is worth noting that when the two offsets are used to determine the "head", the result may not correspond to any sample in the continuous signals due to the sampling rate. Under such circumstance, we try to find the closest time-stamp in the continuous signal. The detailed codes with comments and full Javadoc can be found in [12].

Output operation
The ways of output have been indicated in the modeling stage and need to be implemented in the output operation. A possible example is that the model indicates to output to a certain database. One can then use the information provided by the model to connect to the very database first, and then store the synchronized signals. This operation needs to be executed every time there is new synchronized data.

5 Conclusion and future work

In this paper, we proposed a new DSL called iDSL for IoT based on MDE. We first start by defining a meta-model with high generality, then create model instances and write OCL restrictions, and then create a XML-like textual syntax called as iDSL. Users can easily define their desired meta-model models through text. After the model has been defined, the model-to-text transformation can be automatically realized by using the M2T plugin we created to convert the model into executable Java code, and then simulation experiment has been performed, including simulation of ECG and EEG device connection, data transmission, synchronization operation and output execution.
It is then proved to be helpful in concrete case in the healthcare field, which mainly focus on the signals synchronization between EEG and ECG devices. The needed codes for different domains can be generated automatically, such as Java code. By doing so, the advantages of Model-Driven Engineering including reusability and verifiability are shown.
In the modeling process, we only have data providers and data processors, but no actuators. However, the role of actuators is very important in some applications. This extension will be consideredfor future work. Another future work is the implementation of formal verification with LOTOS in different perspectives.

References

1. Ahamed, F., Farid, F.: Applying internet of things and machine-learning for personalized healthcare: Issues and challenges. In: 2018 International Conference on Machine Learning and Data Engineering (iCMLDE). pp. 19–21 (2018). https://doi.org/10.1109/iCMLDE.2018.00014
2. Chaabene, S., Bouaziz, B., Boudaya, A., Hökelmann, A., Ammar, A., Chaari, L.: Convolutional neural network for drowsiness detection using eeg signals. Sensors **21**(5) (2021)
3. Ciccozzi, F., Spalazzese, R.: Mde4iot: supporting the internet of things with model-driven engineering. In: International Symposium on Intelligent and Distributed Computing. pp. 67–76. Springer (2016)
4. Geoffroy, G., Chaari, L., Tourneret, J.Y., Wendt, H.: Drowsiness detection using joint EEG-ECG data with deep learning. European Signal Processing Conference EUSIPCO pp. 1–5 (Sept 2021)
5. Gillis, A.S.: What is internet of things (iot)? (2021), `https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT`, last accessed 10 Sept. 2021

6. Harrand, N., Fleurey, F., Morin, B., Husa, K.E.: Thingml: a language and code generation framework for heterogeneous targets. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 125–135 (2016)
7. Kantarci, B., Mouftah, H.T.: Trustworthy sensing for public safety in cloud-centric internet of things. IEEE Internet of Things Journal **1**(4), 360–368 (2014). https://doi.org/10.1109/JIOT.2014.2337886
8. Laplante, P.A., Kassab, M., Laplante, N.L., Voas, J.M.: Building caring healthcare systems in the internet of things. IEEE systems journal **12**(3), 3030–3037 (2017)
9. Nepomuceno, T., Carneiro, T., Maia, P.H., Adnan, M., Nepomuceno, T., Martin, A.: Autoiot: a framework based on user-driven mde for generating iot applications. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. pp. 719–728 (2020)
10. OCL: Homepage (2021), `https://www.omg.org/spec/OCL/`, last accessed 16 Sept. 2021
11. Pramudianto, F., Indra, I.R., Jarke, M.: Model driven development for internet of things application prototyping. In: SEKE. pp. 703–708 (2013)
12. Qiyue Fan, Q.M.: idsl source-code (2021), `https://github.com/JulienFan/iDSL`, last accessed 10 Sept. 2021
13. Salihbegovic, A., Eterovic, T., Kaljic, E., Ribic, S.: Design of a domain specific language and ide for internet of things applications. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 996–1001. IEEE (2015)
14. Shaikh, Y., Parvati, V.K., Biradar, S.R.: Survey of smart healthcare systems using internet of things (iot) : (invited paper). In: 2018 International Conference on Communication, Computing and Internet of Things (IC3IoT). pp. 508–513 (2018). https://doi.org/10.1109/IC3IoT.2018.8668128
15. Sirius: Homepage (2021), `https://www.eclipse.org/sirius`, last accessed 10 Sept. 2021
16. Wu, F., Miao, Z., He, C.: Remote monitoring system for intelligent slaughter production line based on internet of things and cloud platform. In: 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan). pp. 538–542 (2020). https://doi.org/10.1109/PHM-Jinan48558.2020.00104