FloWare: An Approach for IoT Support and Application Development

Flavio Corradini, Arianna Fedeli, Fabrizio Fornari(✉), Andrea Polini, and Barbara Re

Computer Science Division, University of Camerino, Camerino, Italy
{flavio.corradini,arianna.fedeli,fabrizio.fornari, andrea.polini,barbara.re}@unicam.it

Abstract. With the advancement of computing technology, we are witnessing the dawn of a new era of the Internet of Things (IoT) paradigm in which objects equipped with sensors, actuators and processing capabilities communicate with each other to serve a given goal. The IoT's intrinsic nature, which uses heterogeneous devices, resources and different communication protocols, complicates IoT applications' design, development, and validation. Reducing the complexity of building IoT applications is one of the current challenges in this area.

To address this challenge, we focus on a model-driven approach to support IoT systems' management and the development of IoT applications. In particular, we propose the FloWare approach and its toolchain, which combine Software Product Line and Flow-Based Programming paradigms to manage the complexity in the various stages of the IoT application development process. An automatic transformation procedure generates the final IoT application, an executable Node-RED flow, starting from a configuration of the designed Feature Models.

Keywords: IoT · Model-driven · Feature model · Flow-based programming · Node-RED

1 Introduction

The Internet of Things (IoT) is a paradigm that gained ground in the scenario of modern wireless telecommunications [3]. The basic idea of this concept is the pervasive presence, all around us, of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators – which, through unique addressing schemes, can interact with each other and cooperate with their neighbours to reach common goals. The development of IoT applications is complex and demanding activities that require the investment of resources and carry a significant risk of failure [14].

One of the main issue recognised in the state of the art [18] is the lack of a software engineering methodology to support the entire IoT applications development life-cycle. This gap results in difficulties in design and developing applications that can be addressed through a Model-Driven Engineering

A. Augusto et al. (Eds.): BPMDS 2021/EMMSAD 2021, LNBIP 421, pp. 350–365, 2021.
https://doi.org/10.1007/978-3-030-79186-5_23

(MDE) approach [26]. The MDE methodology allows to automatically generate easily maintainable and better-quality software based on modelled system requirements. This methodology can foster software productivity in IoT scenarios supporting better solutions, reducing development time and costs, and increasing its quality [18]. In MDE methodology, different models can be designed to describe IoT system requirements and functionalities. Models gaining attention in IoT scenarios have been consolidated in the academy and industry under the Software Product Line (SPL) paradigm. The SPL paradigm refers to software engineering methods for creating a collection of similar software systems from a common model [20]. One of the most applied SPL models in IoT scenarios is the Feature Model [10], which is used to model all possible points of variability and commonality of IoT domains [5]. Feature Models can be used to derive documents, other models, or chunks of code that the developer can arrange for defining a software application.

Referring to the code and programming of IoT applications, the increasing complexity of application development in the IoT domain is well handled by applying the Flow-Based Programming (FBP) paradigm [27]. The FBP is a type of data-flow programming mainly focused on data and their manipulation, differing from other techniques such as workflow-based in which the focus is on representing activities and their execution order [15]. The FBP suits well in the context of the IoT, where the pillar is the reception, manipulation and sending of data as it sees the application as a set of "black-box" software processes; each process has a specific behaviour and exchange data through predefined connections. Thanks to visual tools based on this programming paradigm, it is possible to easily connect different processes to form different applications without changing their internal content [23].

To reduce the complexity and facilitate all the various stages of IoT applications development, we propose *FloWare*, an MDE approach that combines Software Product Line and Flow-Based Programming paradigms. Our approach supports the IoT applications development from the design to software development, including different phases, actors, steps, and artefacts that allow realising model-to-code transformation. We also propose a toolchain and an open-source core component, named *FloWare Core*[1], to support the approach. Finally we describe how FloWare can be applied to a realistic scenario concerning the management of IoT devices and the development of IoT applications in a Smart Campus.

The paper is organised as follows. Section 2 gives an insight on related works concerning the modelling and development of IoT applications based on Software Product Line and Flow-Based Programming paradigms. Section 3 describes the *FloWare* approach for modelling and developing IoT applications. Section 4 presents an introduction to our supporting toolchain. Section 5 reports on the application of *FloWare* in practice. Section 6 concludes the paper by touching upon directions for future work.

[1] FloWare Core: http://pros.unicam.it/floware/.

2 Related Work

Nowadays, the combination of Flow-Based Programming and Software Product Line paradigms is not a consolidated trend concerning IoT applications' development, but it presents a great potential [22].

For what concerns the use of FBP in different IoT domains, examples from [6,7,11,21,24] shows its adaptability in small-medium cases like Smart City, Smart Home, Smart Building, Smart Laboratory. In [12,13,25] are presented applications on large scenarios like Smart Logistic, Smart Hospitals and Smart Military Environments. The literature mentioned above highlights the use of FBP to facilitate the interconnection between heterogeneous devices using different technologies. In particular, we found a common usage of Node-RED[2], an FBP tool that gained research and industry attention over the last years. Node-RED allows the interconnection of devices and services to develop IoT applications. Thanks to the availability of a visual editor, Node-RED enables the easy creation of IoT applications through the interconnection of asynchronous components isolated from the application context.

Although many research works focus on IoT application development, according to the literature [23] emerges a strong need for standardisation. Currently, no standard approach describes the process of developing IoT applications based on this paradigm. The developer does not have a guiding model to follow during IoT applications; this slows the entire application development process and exposes the developer to introduce errors. Furthermore, the developer must deal with the functional specifications of the devices (e.g., the way the device communicates, the application protocol they use, the port numbers) to correctly manage the devices communications and produced data. It directs away from the developer attention from the actual development of the application. Possible support to help a developer designing and developing an IoT application can come from using a Model-Driven Engineering (MDE) methodology. The use of MDE methodology in the software development process is gaining more attention thanks to a high level of abstraction using models as a base for creating the software [9].

The use of Feature Models of the Software Product Line paradigm presents positive evidence for the adaptation in the context of IoT [10]. In [29] and [17], Feature Models are used to model complete heterogeneous systems like Wireless Sensor Networks and Body Area Networks. In [2,19], different authors design the same Smart Home scenario using different levels of abstraction and focusing on different functional requirements. In [16] and [1], Feature Models represent the design configuration of a single IoT device or a set of devices and their communication protocols without linking them to a specific IoT application domain. In [8], the scope is to manage the evolution of a family of middleware for smart environments using cardinality-based Feature Models to express the structural variability of all the involved devices, applications, network protocols. In [28], Feature Models are used to support the representation of a Body-Area

[2] Node-RED: https://nodered.org/.

Network application to analyse functional and non-functional requirements in IoT-oriented healthcare applications. In [4], the authors propose a model-based reconfiguration engine, that uses Feature Models, for dynamically reconfiguring IoT system architectures. The paper presents a Smart Home example scenario to demonstrate how an autonomic reconfiguration can be achieved using Feature Models at run-time.

Although previous works focus on the use of Feature Models for the IoT, most of them apply the models mainly to describe the reference domain of their IoT system. In our approach, we use Feature Models to design the IoT domain and support and guide the developers/users in the software development, using the IoT system configurations derived from the models to produce code that automatically allows the devices' interconnection and data visualisation.

It is for filling the gaps that emerged from the literature that we present in the following sections our *FloWare* approach, which combines Feature Models and Flow-Based Programming to provide an MDE methodology for facilitating the development of IoT applications starting from the modelling of IoT systems and devices.

3 The FloWare Approach

In this section, we present our *FloWare* approach, which supports the modelling and development of IoT applications. We describe the various *phases*, *actors*, *steps* and *artefacts* that characterise our approach, as depicted in Fig. 1.

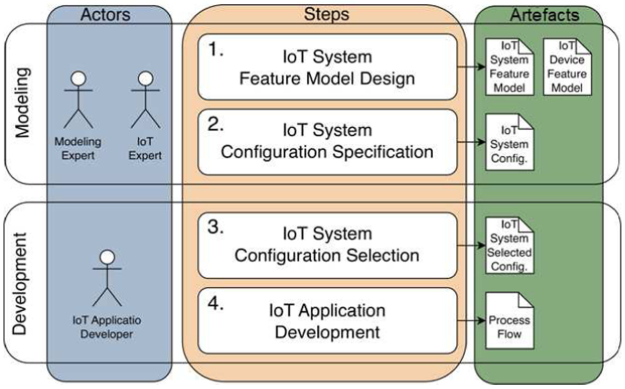

Actors
Steps
Artefacts
Modeling
Development
Modeling Expert
IoT Expert
IoT Applicatio Developer
1. IoT System Feature Model Design
2. IoT System Configuration Specification
3. IoT System Configuration Selection
4. IoT Application Development
IoT System Feature Model
IoT Device Feature Model
IoT System Config.
IoT System Selected Config.
Process Flow

Fig. 1. FloWare approach

Phases. The approach is divided into two main phases: the *Modelling* and the *Development* phases. The modelling phase involves the design and configuration of Feature Models to represent the entire IoT domain. The development phase involves the automatic transformation of models into code to develop the actual IoT application.

Actors. As described in [18], IoT application development is a multidisciplinary process that intersects heterogeneous knowledge from the different involved actors. These actors have different roles in the development process. In our approach, in the modelling phase, we require the involvement of human actors such as a *Modelling Expert* (*ME*), an expert capable of designing and representing specific domains using modelling languages and tools, and an *IoT Expert* (*IoTE*), an expert of the Internet of Things domain responsible for the management of IoT devices deployed (or to deploy) in the IoT system. Instead, the development phase requires an *IoT application developer* to exploit the potential provided by the approach to develop IoT applications.

In the following, the various steps and artefacts (respectively indicated in bold and italic) involved in the approach are described in details.

IoT System Feature Model Design. The ME and the IoTE are in charge of defining two Feature Models to represent the entire scenario (Step 1 of Fig. 1): the *IoT System Feature Model* and the *IoT Device Feature Model*. The required Feature Models are designed based on the ME and IoTE expertise, or they can use already deployed models saved in a common repository. The IoT system Feature Model provides an overview of an IoT system's generic domain and all functionalities it could present; the second provides an overview of IoT devices' characteristics. Figure 2 reports examples of the two models that can be defined to represent a simple IoT system and device. The experts can start the modelling activities from a model presenting a similar skeleton and then modify features and relations according to the modelled IoT system and devices' needs.

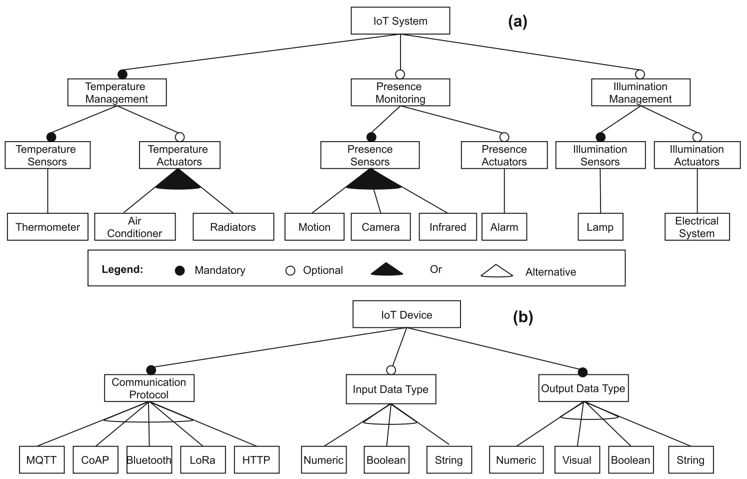


Fig. 2. (a): IoT system feature model; (b): IoT device feature model

For the IoT system Feature Model, presented in Fig. 2(a), the tree's root, named IoT System, will be substituted with a more appropriate term to represent the scenario to be specified (e.g., Smart University). Then, different features linked to the root can be represented to describe the various functionalities that the considered IoT System can have. Some of these functionalities are linked by a mandatory link, so they must be selected in the configuration phase; others are optional and can be inserted into the configuration according to appropriate needs. For each functionality inserted, it is necessary to represent all the devices involved in the scenario, dividing them between sensors and actuators. For the IoT devices Feature Model shown in Fig. 2(b), one of the primary information to be inserted regards the communication protocols used to interact with the devices. Other information necessary to allow the correct manipulation of the devices' data refers to the data types that the device can handle. Regarding the data that the device sends or receives, it is mandatory to define if they are numeric values, Boolean, etc., to allow the system to process them correctly.

IoT System Configuration Specification. The designed Feature Models will be used as a starting point to specify the *IoT System configuration* in a scenario. The configurations based on the Feature Models can be related to deployed IoT systems or under-deployment systems. IoT configurations represent different instances of the same model and are usually different based on several factors depending on how the IoT system is deployed. For example, referring to Fig. 2(a), the temperature management functionality could be realised in one location using temperature sensors and an air conditioning system, while in another site through the use of radiators. Concurrently, other IoT systems may need to implement features such as presence monitoring rather than illumination management to meet the use cases needs. This differentiation produces different configurations starting from the same model.

An example of a valid IoT System configuration is the one reported in Fig. 3(a), where the IoT System is composed of two functionalities, one related to the temperature management, as a mandatory functionality to represent the configuration, and the presence monitoring, with their corresponding devices.

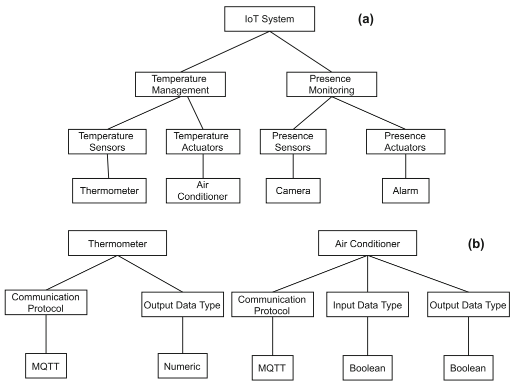


Fig. 3. (a): IoT system configuration; (b): different IoT devices configurations

The features chosen in the IoT system configuration include the use of devices divided between sensors and actuators, which need to be configured. Following the two models in Fig. 3(b), we can see how the two devices use the same protocols to communicate but provide different types of data to be processed. In particular, if the device is an actuator, it also includes the input data type that specifies the data type to receive as input.

In addition, the ME and the IoTE can specify where IoT devices are deployed (building name, room name, geolocation, etc.) together with other specific information according to each device's chosen protocol. Devices related information can include: an identifier, stored together with its location (e.g., GPS coordinates and a known nomenclature for defining room names), an MQTT broker address, and a topic (UTF-8 string) used to access the device data. The defined IoT System Configuration with all specified information will then be made available to the IoT Application developer that desires to develop an IoT application for one or more IoT systems.

IoT System Configuration Selection. It allows an IoT Application Developer to choose which IoT system configuration to interact with. Starting from a configuration like the one reported in Fig. 3(a), the IoT application developer can choose which functionality of the deployed system to access together with the respective IoT devices. This choice will be guided by the developer requirements for the specific application and the information stored, which allows distinguishing between devices of the same type (e.g., the position or the type of device could be a selection factor). For example, referring to the configuration in Fig. 3(a), an IoT developer can build an application using only the Temperature Management functionalities and access the available thermometers' information and command the air conditioner instead of selecting the entire configuration. The *IoT System selected configuration* obtained will be used to generate an IoT application.

IoT Application Development. After the IoT developer specified which functionality of an IoT system and which devices to interact with, the environment for developing the IoT application is configured accordingly. For instance, if the developer specified the need to interact with a temperature sensor that uses an MQTT protocol to communicate, the environment is automatically configured to include the MQTT functionality. In addition, an IoT application is automatically generated in the form of a *process flow* that communicates with the device and shows the obtained values through a dashboard.

4 The Supporting Toolchain

To support the usage of our *FloWare* approach, we defined a toolchain and developed an open-source component named *FloWare Core*. The FloWare Core component is an open-source JavaScript software that can be installed both on the cloud and on less powerful machines such as gateways (for example, raspberry etc.). Thanks to a graphical and intuitive interface, it is possible to easily configure all the IoT systems and devices and to develop IoT applications based on them. Specific information regarding FloWare Core, how to access its source code and set it up, is reported on http://pros.unicam.it/floware/. In the following, we provide a technical description of the toolchain components and we also describe how to use them.

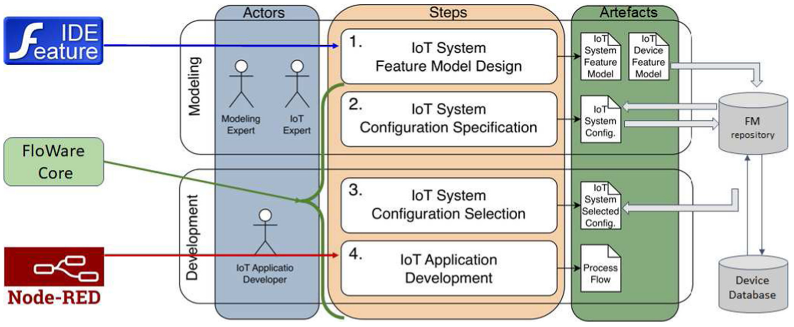


Fig. 4. FloWare approach with the supporting toolchain

As explained in Sect. 3, initially, the IoTE and ME will design the two models. The first model represents the reference IoT system, and the other represents all the IoT devices' functionalities (communication protocol, type of data to be processed, etc.). For the design of the Feature Models, as shown in *Step 1* of Fig. 4, we suggest using Eclipse FeatureIDE[3], which allows to generate a Feature Model and verify its validity. Once the Feature Models have been designed, they will be saved in a Feature Model repository as XML files, as shown in Fig. 4. These models will be used as input for FloWare Core to support the following phases of the approach.

As reported in Fig. 5, the FloWare Core's architecture is formed by several components; each performs a specific task to support the configuration and the automatic generation of the IoT application. In the following, we provide a detailed description of each component.

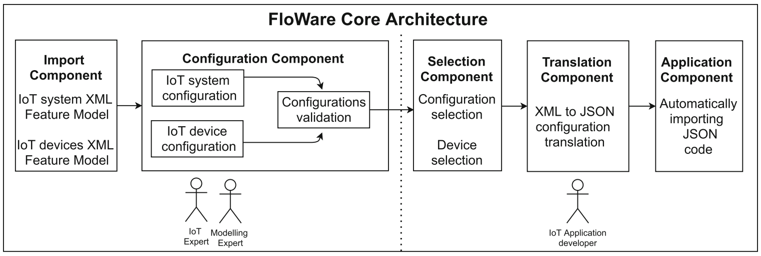


Fig. 5. The FloWare Core architecture. The left side reports components and actors involved in the first phase of the approach. The right side reports components and actors involved in the second phase.

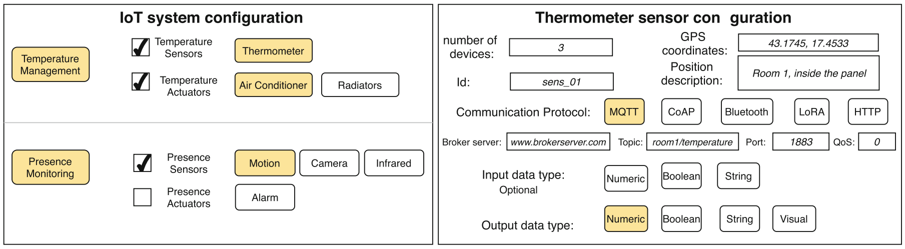
The *Import Component* manages the import of the XML files representing the designed Feature Models. These models serve as a basis to support the configuration of the IoT system and the devices involved. FloWare Core comes with

[3] FeatureIDE: http://www.featureide.com/.

some Feature Model templates that can be used as a starting point for modelling the IoT system and the IoT devices avoiding the complete redesign phase. Once the XML Feature Models are imported, the Configuration Component provides a graphical interface to help the experts in the system configuration and the addition of information regarding the devices involved.

A prototype of the *Configuration Component*, represented in Fig. 6, reports an extract of an IoT system's possible configuration. From this configuration section, the experts can select all the functionalities and devices that will be deployed on the IoT system. Then, for each chosen device, they are requested to fill a form concerning the specific information of each device involved. A user's configuration is valid if it ensures compliance with the Feature Model designed during the first step of the approach and elaborated by the tool. Once the systems and devices' configurations are completed, they will be saved in the repository, while the devices information will be saved in a device database. Both entities communicate to keep track of the systems and related devices involved in the scenario. In particular, for what concerns the devices, all the information are saved following the WoT Standard[4].

Once terminated the devices and systems' modelling phase, the IoT Application developer can use the FloWare Core to support IoT applications' development.


IoT system configuration

- Temperature Management
  - [✓] Temperature Sensors: [✓ Thermometer]
  - [✓] Temperature Actuators: [✓ Air Conditioner] Radiators
- Presence Monitoring
  - [✓] Presence Sensors: [✓ Motion] Camera Infrared
  - [ ] Presence Actuators: Alarm

Thermometer sensor con guration

number of devices: 3
GPS coordinates: 43.1745, 17.4533
Id: sens_01
Position description: Room 1, inside the panel
Communication Protocol: [✓ MQTT] CoAP Bluetooth LoRA HTTP
Broker server: www.brokerserver.com Topic: room1/temperature Port: 1883 QoS: 0
Input data type: Optional Numeric Boolean String
Output data type: [✓ Numeric] Boolean String Visual

Fig. 6. Prototype of a configuration of an IoT system and a device configuration prototypes using FloWare Core

In the *Selection Component*, the developer selects one of the previously configured systems and specifies which functionalities of the system want to represent in the application. From all the devices involved in the selected scenario, the developer chooses the ones to include in the IoT application. In this way, we leave the developer free to realise the IoT application according to specific needs. For example, the developer may need to represent only a particular room in a building; thus, he/she have the possibility to freely select only the appropriate devices from the entire configuration to represent the desired scenario.

[4] WoT Standard: https://www.w3.org/TR/wot-thing-description.

Then, the *Translation Component* take as input the functionalities and devices selected in XML format and automatically encode them in a JSON file that represents the IoT application. The JSON file is generated to be processable by Node-RED, a development tool incorporated into FloWare Core, to provide a single working and processing solution. Node-RED allows the composition of IoT applications using components; each performs a specific operation and retrieves a result. It also offers the possibility to use some dashboard components to visualise the data obtained from the devices in real-time. In our approach, an IoT application is represented as a Node-RED process flow written in the JSON file. In the generated file, each JSON field represents a different Node-RED component that performs a specific function.

```
<configuration>
<feature automatic="selected"  name="Temperature"/>
<feature automatic="selected"  name="Communication protocol"/>
<feature automatic="undefined" name="MQTT"/>
<feature automatic="undefined" server="www.brokerserver.com"/>
<feature automatic="undefined" port="1883"/>
<feature automatic="undefined" qos="0"/>
<feature automatic="undefined" topic="room1/temperature"/>
<feature automatic="undefined" name="Output Data type"/>
<feature automatic="undefined" name="Numeric"/>
</configuration>
```

```
{ id: 42a56979.773094,
broker: www.brokerserver.com,
port: 1883,
name: Temperature,
qos: 0,
topic: room1/temperature,
type: mqtt input }
```

Fig. 7. Automatic translation from an XML device configuration to JSON component

Figure 7 shows the correspondence between the information collected in the XML configuration and the translation into a JSON field. The conversion in example allows filling each field of the JSON file with the information necessary to start the MQTT communication with the device. There are also generated specific JSON fields able to process the data types received, such as the Numeric type in the example, and send them to other fields that show them in a graphical form. The JSON file also reports the connections between each field, intending to generate a process flow that brings the information from the beginning component (e.g., the component of the receiving device) to the final one (e.g., the data display component).

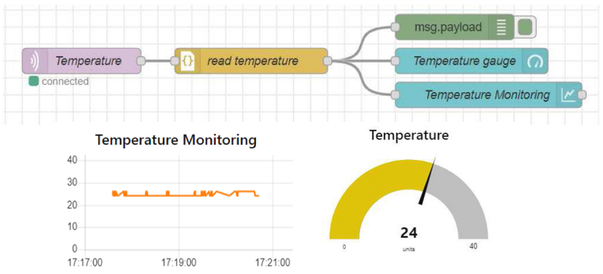


Fig. 8. Temperature monitoring application with dashboard

For each involved device, we obtain a process flow generated automatically by the FloWare Core component, like the one reported in Fig. 8. The generated flow is automatically imported by the *Application Component* into Node-RED, giving the user the possibility to add other functionalities to the application or immediately execute it to inspect the device's data in real-time in a dashboard. In this way, we obtain an IoT application with all the device specifications automatically imported from the previously defined XML configurations. The developer is not obliged to know and insert specific information regarding each device, but the FloWare Core automatically retrieve them from the Feature Model repository and the device database. Then, the developer can expand the generated application by inserting additional Node-RED components.

5 Case Study

In this section, we present a realistic case study to show how our *FloWare* approach can support the management of IoT systems in a Smart Campus, from the models' domain design to the development of a temperature management IoT application.

A university would like to standardise the different IoT systems deployed over the years through the various departments to have a clear vision of the practices adopted. At the same time, the university wants to reason about improvements in terms of new IoT systems that could be deployed in other departments and the development of IoT applications that could take advantage of all the sensors and actuators that have been deployed. Applying *FloWare* in such a scenario, could provide support in the construction of IoT systems and the decoupling of IoT system configuration from IoT application development for the developers that manage these systems. In this way, aim at reducing complexity for the developer, who can work with already configured systems and devices. We can assume that all the university departments are equipped with IoT devices to achieve basic intelligent systems (i.e., temperature and illumination control systems). At the same time, they may include different IoT systems based on the department's specific necessity. In the following, we report some examples of what specific departments need.

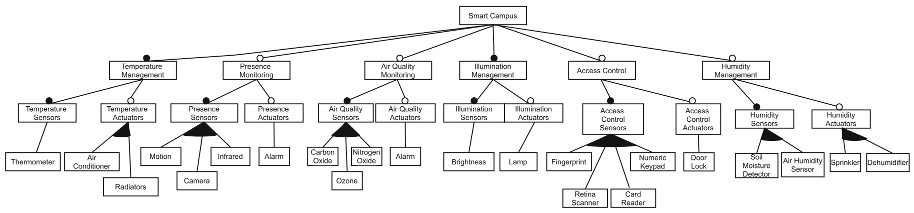


Fig. 9. Feature Model representing the entire Smart Campus functionalities

Inside the *Chemistry department*, for safety reasons, it is mandatory to have a system for air-quality monitoring that can immediately report the levels of CO, CO_2, SO_2 and NO_2 that may derive from experiments conducted in the laboratories. In the *Biology department*, there is the necessity of monitoring experiments involving plants. In particular, there is the necessity to have a system able to control and act on the soil's humidity level and optimally manage the various plants' level of light and temperature. In the *Computer Science department*, on the other hand, there is the need to control accesses to the different server rooms located in it. These rooms can only be opened by authorised staff who can use various access systems, including fingerprint or retina scanner, identification badges.

Using our approach, it is possible to model the entire IoT Smart Campus scenario, as reported in Fig. 9. In particular, are highlighted all the different functionalities and devices that may be made available in the departments. It is important to note that temperature and illumination management are mandatory fields in each configuration as defined at the beginning of the case study; the others are optional. In the same way, IoT devices have to be modelled. The experts can design a model like the one previously reported in Fig. 2(b) or use the FloWare Core's templates.

After designing the Feature Models, the experts use the graphical interface provided by our FloWare Core component to configure all the various devices to use in each department. We defined, using our FloWare Core component, some configurations to represent specific departments' needs in a Feature Models form, as shown in Fig. 10.

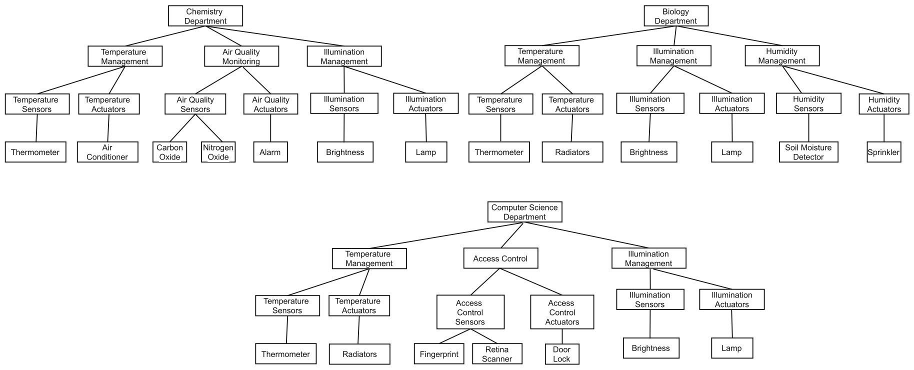


Fig. 10. Departments configuration starting from the IoT system Feature Model

At this point, all the departments' configurations are saved in an Feature Model repository. Starting from these final configurations, we decided to develop an IoT application to test our *FloWare* approach and the FloWare Core component. Our component was installed in a Raspberry Pi to be used as a gateway; thus, it collects the data it receives from the devices and shows them in real-time. Using the FloWare Core's graphical interface, we selected which department, systems, and devices to use in the development of the IoT application, as shown in Fig. 11.

We limited our application to the temperature functionality present in the Chemistry Department' configuration. Then, we selected the devices that we want to automatically include in the application development for that department. In this way, we did not necessarily have to be aware of every device's specifications from which to get information. We only needed to know the functionalities of a device exposed (e.g. temperature measurement) and its location (e.g. place description or GPS coordinates) to identify and choose it uniquely.

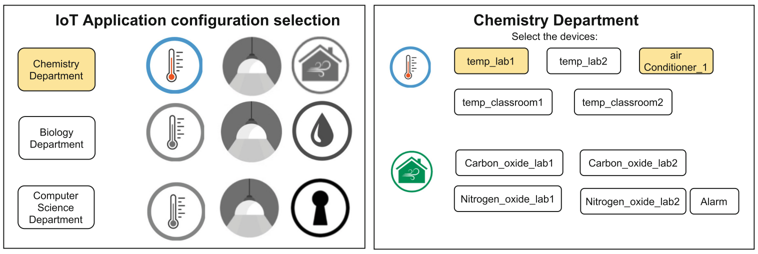


Fig. 11. A prototype of departments and respective devices selection using FloWare Core component

According to the selection, for each chosen device, the FloWare Core component generated an application like the one reported in Fig. 8. In this way, the application can automatically read and show the devices' data incoming in real-time and, if needed, send data to interact with them.

In addition, as shown in Fig. 12, we manually expanded the application to automatise temperature management functionality by adding other components presents in the Node-RED palette, such as a switch conditioner state to automatise the activation of an air-conditioner based on the department's actual temperature. Furthermore, we added an HTTP node to perform an HTTP request to send the data obtained from the devices to a third-party application for external uses.

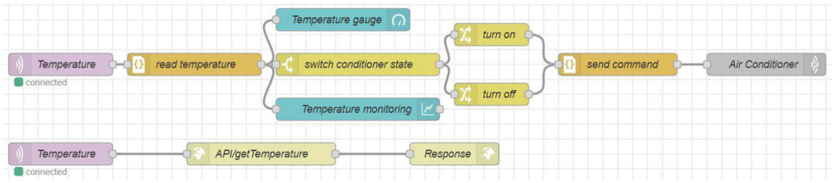


Fig. 12. A complete temperature management IoT application on Node-RED

6 Conclusion

To resolve the complexity of developing IoT applications, we presented the *FloWare* approach, which combines and exploits the potential of the Software Product Lines paradigm, in the form of Feature Models, with the Flow-Based Programming paradigm of the Node-RED tool. Our approach provides a way to facilitate IoT applications development, providing an approach that covers the IoT system's modelling up to its implementation. In particular, we argued that moving the focus on modelling and configuring different IoT systems from experts can help the IoT developers build IoT applications. With our solution, the developers remain unaware of each device's technical specifications because it will inherit them from the experts' previous configurations.

We also presented a supporting toolchain for the *FloWare* approach, focusing on our FloWare Core component. We illustrated its application to a sample case study; the case study involved a Smart Campus. From our studies and tests, the conjunction between the two paradigms seems a suitable choice which requires further studies to consolidate and extend the approach.

As future works, we want to involve other IoT and modelling experts to design and develop heterogeneous IoT systems in practice to validate and test the entire approach on a complete real scenario. Referring to the FloWare Core component, we intend to expand the set of templates including more complex ones to simplify and speed up the development of IoT applications. Those templates may support the developer in setting up interactions with external services for data storage and data analytic.

References

1. Abbas, A., Siddiqui, I.F., Lee, S.U.J., Bashir, A.K.: Binary pattern for nested cardinality constraints for software product line of IoT-based feature models. IEEE Access **5**, 3971–3980 (2017)
2. Alférez, M., Moreira, A., Amaral, V., Araújo, J.: Model-driven requirements specification for software product lines. In: Model-Driven Domain Analysis and Software Development: Architectures and Functions, pp. 369–386. IGI Global (2011)
3. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. Comput. Networks **54**(15), 2787–2805 (2010)
4. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Autonomic computing through reuse of variability models at runtime: the case of smart homes. Computer **42**(10), 37–43 (2009)
5. Cognini, R., Corradini, F., Gnesi, S., Polini, A., Re, B.: Business process flexibility - a systematic literature review with a software systems perspective. Inf. Syst. Front. **20**(2), 343–371 (2016). https://doi.org/10.1007/s10796-016-9678-2
6. Cognini, R., Corradini, F., Polini, A., Re, B.: Extending feature models to express variability in business process models. In: Persson, A., Stirna, J. (eds.) CAiSE 2015. LNBIP, vol. 215, pp. 245–256. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19243-7_24
7. Cognini, R., Corradini, F., Polini, A., Re, B.: Business process feature model: an approach to deal with variability of business processes. In: Domain-Specific Conceptual Modeling, pp. 171–194. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_8
8. Gámez, N., Fuentes, L.: Architectural evolution of FamiWare using cardinality-based feature models. Inf. Softw. Technol. **55**(3), 563–580 (2013)
9. Gascueña, J.M., Navarro, E., Fernández-Caballero, A.: Model-driven engineering techniques for the development of multi-agent systems. Eng. Appl. Artif. Intell. **25**(1), 159–173 (2012)
10. Geraldi, R.T., Reinehr, S.S., Malucelli, A.: Software product line applied to the Internet of Things: a systematic literature review. Inf. Softw. Technol. **124**, 106293 (2020)
11. Havard, N., McGrath, S., Flanagan, C., MacNamee, C.: Smart building based on Internet of Things technology. In: International Conference on Sensing Technology, pp. 278–281 (2018)
12. Jain, R., Tata, S.: Cloud to edge: distributed deployment of process-aware IoT applications. In: International Conference on Edge Computing, pp. 182–189. IEEE Computer Society (2017)
13. Jalaian, B., Gregory, T., Suri, N., Russell, S., Sadler, L., Lee, M.: Evaluating LoRaWAN-based IoT devices for the tactical military environment. In: World Forum on Internet of Things, pp. 124–128. IEEE (2018)
14. Lee, I., Lee, K.: The Internet of Things (IoT): applications, investments, and challenges for enterprises. Bus. Horiz. **58**, 431–440 (2015)
15. Morrison, J.P.: Flow-Based Programming: A New Approach to Application Development, 2nd edn. CreateSpace, Scotts Valley (2010)
16. do Nascimento, N.M., Alencar, P.S.C., Lucena, C., Cowan, D.D.: An IoT analytics embodied agent model based on context-aware machine learning. In: IEEE International Conference on Big Data, pp. 5170–5175. IEEE (2018)
17. Ortiz, Ó., García, A.B., Capilla, R., Bosch, J., Hinchey, M.: Runtime variability for dynamic reconfiguration in wireless sensor network product lines. Int. Softw. Prod. Line Conf. **2**, 143–150 (2012)

18. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. J. Syst. Softw. **103**, 62–84 (2015)
19. Pereira, J.A., Maciel, L., Noronha, T.F., Figueiredo, E.: Heuristic and exact algorithms for product configuration in software product lines. In: International Systems and Software Product Line Conference, p. 247. ACM (2018)
20. Pohl, K., Bockle, G.V.D.L.F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28901-1
21. Poongothai, M., Subramanian, P.M., Rajeswari, A.: Design and implementation of IoT based smart laboratory. In: International Conference on Industrial Engineering and Applications, pp. 169–173. IEEE (2018)
22. Prehofer, C., Chiarabini, L.: From Internet of Things mashups to model-based development. In: 39th Annual Computer Software and Applications Conference, pp. 499–504. IEEE Computer Society (2015)
23. Ray, P.P.: A survey on visual programming languages in Internet of Things. Sci. Program. **2017**, 1231430:1–1231430:6 (2017)
24. Sicari, S., Rizzardi, A., Coen-Porisini, A.: How to evaluate an internet of things system: models, case studies, and real developments. Softw. Pract. Exp. **49**(11), 1663–1685 (2019)
25. Sicari, S., Rizzardi, A., Coen-Porisini, A.: Smart transport and logistics: a node-red implementation. Internet Technol. Lett. **2**(2), 34 (2019)
26. Sosa-Reyna, C.M., Tello-Leal, E., Alabazares, D.L.: Methodology for the model-driven development of service oriented IoT applications. J. Syst. Archit. **90**, 15–22 (2018)
27. Szydlo, T., Brzoza-Woch, R., Sendorek, J., Windak, M., Gniady, C.: Flow-based programming for IoT leveraging fog computing. In: International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 74–79. IEEE Computer Society (2017)
28. Venckauskas, A., Stuikys, V., Jusas, N., Burbaite, R.: Model-driven approach for body area network application development. Sensors **16**(5), 670 (2016)
29. Venckauskas, A., Stuikys, V., Toldinas, J., Jusas, N.: A model-driven framework to develop personalized health monitoring. Symmetry **8**, 65 (2016)