Software and Systems Modeling (2023) 22:131–158
https://doi.org/10.1007/s10270-022-01026-9

SPECIAL SECTION PAPER

FloWare: a model-driven approach fostering reuse and customisation in IoT applications modelling and development

Flavio Corradini[1] · Arianna Fedeli[1] · Fabrizio Fornari[1] · Andrea Polini[1] · Barbara Re[1]

Received: 31 October 2021 / Revised: 13 July 2022 / Accepted: 16 July 2022 / Published online: 6 August 2022

Abstract

The relevance of IoT-based solutions in everyday life is continuously increasing. The capability to sense the world, activate computation based on data gathered by sensors, and possibly produce reactions on the world itself results in an almost never-ending identification of novel IoT solutions and application scenarios. Nonetheless, IoT's intrinsic nature, which includes a high degree of variability in used devices, data formats, resources, and communication protocols, complicates the design, development, reuse and customisation of IoT-based software systems. In addition, customers require personalised solutions strongly based on their specific requirements. Reducing the complexity of building customised solutions and increasing the reusability of developed artefacts are among the topmost challenges for enterprises and IoT application developers. Upon these challenges, we propose a model-driven approach organising the modelling and development of IoT applications in different steps, handling the complexity in representing the IoT domain variability, and empowering the reusability of design decisions and artefacts to simplify the derivation of customised IoT applications. Our proposal is named *FloWare*. It follows the typical path of an MDE solution, providing modelling support through feature models to fully represent and handle the possible variability of devices in a specific IoT application domain. Once a specific configuration has been selected, this will be complemented with specific information about the deployment context to automatically derive fragments of the IoT applications, that will be successively combined by the developer within a low-code development environment. The approach is fully supported by a toolchain that has been released for public use.

Keywords IoT application development · Model-driven · Variability modelling · Low-code · Customised applications · Design artefact reusability

1 Introduction

The Internet of Things (IoT) is a paradigm that gained ground in the context of modern wireless telecommunications [6]. The basic idea stands in the pervasive presence, all around us, of a variety of things or devices, 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 possibly reach common goals [17]. IoT applications are developed to handle the interaction among many of these devices and the physical environment to produce valuable services to users. Therefore, an IoT application generally includes a computational part that can be distributed over the devices and connected to cloud-based infrastructure. This logical component embeds the logic that permits the delivery of the intended service. For example, the automatic management of lighting in a building

Communicated by Iris Reinhartz-Berger, Jelena Zdravkovic, Asif Gill.

✉ Fabrizio Fornari
fabrizio.fornari@unicam.it

Flavio Corradini
flavio.corradini@unicam.it

Arianna Fedeli
arianna.fedeli@unicam.it

Andrea Polini
andrea.polini@unicam.it

Barbara Re
barbara.re@unicam.it

[1] School of Science and Technology, Computer Science Department, University of Camerino, Via Madonna delle Carceri, 7, Camerino, Italy

can be achieved through the combined usage and coordination of light sensors, presence sensors, and motorised shutters and lighting systems that can change a room's lighting conditions. In such a case, the collected data from these sensors could flow towards a cloud-based infrastructure where sensor data are collected and fed to a computational component. Such component can then decide when and how to activate the lighting system and the motorised shutters to reach specified objectives.

Developing IoT-based applications is a complex and demanding activity asking for a relevant effort to enterprises and carrying a significant risk of failure [35,53,81,86]. Indeed, the development of an IoT application generally requires addressing many interrelated aspects, starting from low-level details related to the heterogeneous involved sensors and actuators, the interacting protocols, the possible definition of complex data manipulation procedures, and the definition of a high-level logic so to coordinate the various devices to reach the established objectives [31,60]. In addition, enterprises often face the challenge of deriving *customised solutions* that better respond to a client-specific context that requests customisation [27]. These requests are challenging for enterprises that have to manage IoT elements with a significant *variability* (in terms of data, communication protocols, and others) to produce competitive applications inside the market [6]. As a result, the development of an IoT application generally is a "single effort" that hardly produces reusable artefacts. The development of an almost identical application will scarcely have the opportunity to exploit the *reusability* of already developed artefacts if the devices to be used, or their displacement over the environment, are different.

To reduce the overall complexity of IoT applications development, fostering reuse and customisation, we propose *FloWare*, an MDE approach that exploits the advantages derived from the use of feature models to develop IoT applications for low-code development environments. *FloWare* provides support from the design to the development and deployment of customised IoT applications. The proposals foresee different phases, actors, steps, and artefacts resulting in a model-to-code transformation approach. The general idea behind *FloWare* is to provide modelling mechanisms that permit the description of general aspects related to an IoT application, and that are not connected to a specific deployment context. Provided abstract models then play the role of a platform-independent model (PIM), permitting the representation of crosscutting concerns about the IoT application under development, and then the representation of knowledge, that can be more easily reused to develop customised IoT solutions. The approach provides configuration and refinement mechanisms that enable deriving models connected to a given deployment context and then to define fully functional solutions.

The entire modelling approach is supported by the ADOxx metamodelling platform[1] for which we developed a dedicated feature models library. The library has been used to support the design of feature models concerning an IoT domain and the possible IoT devices involved. Defined feature models can be configured by inserting specific information and a suitable translator that automatically derives IoT application artefacts from specified configurations. We also developed the open-source FloWare platform[2] to validate and support the proposed IoT application development. Overall, the result can be considered a low-code development environment to develop, customise, and deploy IoT software applications.

Outline The *FloWare* approach, and consequently the paper, follows the design science research method (DSRM) [37,65], a qualitative research approach focused on the design process, and generating knowledge about the method used to design an artefact, and the artefact itself. An initial version of the approach has been already, and shortly, introduced in [23]. This paper extends the previous version in many different dimensions and includes a more thoughtful comparison with emerging approaches proposed in the literature. In particular, the paper includes details on the *FloWare* toolchain and better clarifies the possible benefits coming from the adoption of the approach through a realistic scenario.

The paper structure is the following. In Sect. 2, the first DSRM step requires defining the IoT application development research problems and gaps derived from the literature, that then justifies the motivation behind the proposed approach. In Sect. 3, the second DSRM step scouts the leading solutions supported by the literature that aims to develop IoT applications, including a discussion about their limitations. Above the design and development DSRM step, described in Sect. 4, we built our *FloWare*, an MDE approach to model and develop IoT applications. In the demonstration step, in Sect. 5, we show the applicability of our approach in a complex smart campus case study, to illustrate how effectively it solves the challenges mentioned above. In the evaluation step, in Sect. 6, we observe and measure how *FloWare* can support the emerged challenges by comparing the objectives of the solution concerning already existing approaches. In Sect. 7, the limitations concerning the developed approach are highlighted, and in Sect. 8, possible future directions are described.

2 Motivation and challenges

The term IoT already emerged more than 20 years ago (the term was coined in 1999 by Kevin Ashton); nonetheless,

[1] ADOxx: https://www.adoxx.org/live/home.

[2] FloWare Platform:https://github.com/PROSLab/FloWare-Core.

according to the literature, there are still many challenges, both from the methodological and the technological side, that makes the development of IoT application a complex endeavour [17]. On the methodological side in [52,79], the authors highlight the lack of software engineering methodologies to decrease the overall complexity of the entire IoT application development. Compared to traditional software development methodologies, building an IoT application has many peculiarities and results in higher complexity, as noted in [82]. Indeed, also [18] reports that there is a strong need for solutions that allow lowering the entry barrier in the development of IoT applications. This is necessary to encourage experts to build solutions to meet the real needs of consumers and to propose innovative ideas in a reasonable time and cost.

The variability of the deployment context is another relevant source of complexity. This clearly impacts the direction of the definition of a development methodology that has to provide tools to handle such a variability. Nonetheless, it is firmly rooted in technological aspects [32,68,79]. In particular, the heterogeneity of the devices needed to sense and manipulate the environment makes an IoT application challenging to adapt to different deployment conditions. Another distinctive peculiarity in the development of IoT applications is that the information needed and manipulated by the computational layer can be produced by heterogeneous data sources in a different context. For instance, the presence of a person in a room could be derived in a given deployment context from the data provided by a motion sensor, while in another deployment context this information could be derived from an access control system. As a result, one of the challenges that IoT solution providers have to face has to do with the production of IoT applications with a high degree of **customisation**. Indeed, adopting development approaches based on the customisation and configuration of general solutions permits the management of different customer needs and deployment contexts, while maintaining adequate costs and times to stay competitive in the market [26].

Focusing on the variability of available devices, many dimensions can be considered in developing a specific IoT application [39,51]. Among the various heterogeneity dimensions, the one related to the communication protocols used by the devices is particularly relevant. Differences can refer to the multiple layers of the communication stack. So we have devices connecting through WiFi, Bluetooth, 5G, or RFID, where each type of communication protocol has its peculiarities (e.g. maximum connection distance, message sending/receiving speed, sending frequency and many others), and devices supporting application protocols such as MQTT, HTTP, CoAP, or proprietary protocols such as LoRaWAN [1]. Having IoT applications in which the involved devices use different protocols can be challenging and expensive. Indeed, in the general case, the IoT application developers will have to handle such a heterogeneity aspect explicitly.

The target domain for IoT applications (e.g. smart home, smart hospital, smart agriculture) generally influences the functionalities typically foreseen by an IoT application to be deployed. Each domain has its peculiarities and asks for a multitude of services that are somehow domain-specific [79]. For example, a smart hospital generally provides wellness, access control in the rooms, optimises temperature and light management, and heart-rate control. Another different domain, such as smart agriculture, can offer services regarding monitoring and acting on soils and plants. Different customers can require various devices to operate, usually to sense or change the physical environment, depending on their needs. In the same way, a high degree of variability concerning required functionalities inside an IoT application is present. Besides the main functionalities necessary inside a specific domain, the resulting IoT application is strongly based on the distinct functionalities required by each customer and the domain of use.

As also observed in [27], the immediate result of the challenges described so far is that IoT applications are often re-written for each deployment context that shows some difference from the initial one. This is detrimental in terms of time and costs for enterprises, which have to develop each new request from scratch, even if the application requirements are similar [3]. In addition to this, as observed in [40], a lack of systematic support for **reusability** does not allow enterprises to build customised IoT applications with reasonable costs and time to market [21]. Moreover, [43] suggests that effective reusability mechanisms could improve the IoT application development efficiency. Reusability mechanisms are based on the fact that once a given structure has been developed or a given knowledge acquired, it must be saved for subsequent reuse. Providing such a mechanism to crystallise the acquired knowledge, and to foster its possible reuse, could be a viable solution that needs more attention. Moreover, software reuse is a valuable option to support, as it allows for better-quality software and increases maintainability.

From the analysis of the reported challenges in the development of IoT applications, the following research questions seemed particularly interesting to us:

RQ1—How do available solutions support the modelling and development of IoT applications to foster reuse and customisation?
RQ2—How can we improve support for customisation and reusability, of IoT application modelling and development?

RQ3—Which are the benefits of *FloWare* compared to available MDE approaches?

The three questions clearly drove the research work we performed and led us to the definition of the *FloWare* approach. The approach follows the model-driven engineering (MDE) paradigm to address the mentioned challenges [77], so to define a methodology and a modelling method that foster reusability and customisation.

3 Related work on solutions for IoT application modelling and development

This section intends to provide an answer to the first research question: "(RQ1)—How do available solutions support the modelling and development of IoT applications to foster reuse and customisation?"

In the following we cluster those proposals found in the literature that somehow relate to solutions for modelling and development of IoT applications. Successively we discuss these proposals on the base of how they provide support to *customisation* and *reusability*, also in consideration of the *variability* factor mentioned in the previous section.

Workflow management systems The adoption of workflow management system for the definition and execution of business processes related to the IoT domain has been a quite flourishing topic, in the last years [22]. In this sense, different notations can be used to represent a workflow and to execute it on an IoT-aware workflow management system. One of the most used notation is the business process model and notation (BPMN). However, this integration poses challenges both to the research community and the industry, that have to clarify how business process models can be made "IoT-aware" [49]. A relevant effort has been conducted in [36,56,84] to increase BPMN expressiveness so to permit capturing and expressing IoT aspects, while others approaches [5,14,80] aim to provide a mechanism to execute these enhanced models. From a modelling perspective, those research works perform an attempt to incorporate IoT elements (e.g. an IoT sensor) inside the BPMN model, including their related information. More in general, all the works describe the usage of BPMN to express business processes including IoT aspects mainly allowing process modelling from a higher perspective. However, most of the approaches do not provide support for executing these enhanced BPMN models.

Approaches based on BPMN allow handling the complexity of representing IoT-aware business processes, and they can provide a certain degree of customisation on the defined models. However, being BPMN a general-purpose modelling language, approaches based on such modelling notation may result in a limited expressiveness concerning peculiar IoT concepts. Indeed, the BPMN language allows the expression of concepts related to the IoT, but with a degree of abstraction that could only include rudimentary support in expressing essential aspects of the IoT domain. Indeed, as emerged from [57], although several works provide efforts to include IoT elements within a BPMN model, at the moment, no standard or common representation of almost all IoT aspects is present. To overcome this problem, it would be necessary to include, validate and use an extended BPMN notation for the IoT domain, which can thus fully exploit its characteristics. The current integration efforts [41,80,81] allows modelling an entire business process by including the concept of devices, their actions and in some cases, even details on their connectivity, to already existing BPMN elements (e.g. activities, events and tasks). However, much effort is still needed to integrate all the relevant concepts of the IoT domain (e.g. the possibility to represent a device behaviour, and that of the entire IoT application) into workflow management systems. In addition, it also emerges from the literature related to IoT-aware business processes a lack of a common engine to be used to execute models defined using an "IoT enhanced" BPMN modelling notation.

Model-driven engineering approaches The use of a model-driven engineering (MDE) approach in developing IoT software products is gaining attention since it permits to abstract technical details through models that can be successively used as a base for further refinement, and to possibly derive executable artefacts [34]. MDE approaches can be structured to provide a clear separation of concerns through the definition and description of different system perspectives, helping in handling the complexity of an IoT application [63]. Adopting an MDE approach enables defining a method for the automatic generation of maintainable and better-quality software based on modelled system requirements. This approach can foster software productivity in IoT scenarios, reducing development time and costs [64].

Compared to traditional information systems development, where the central pillar for the developer is coding, in an MDE approach [72], the developer puts models as the basis of the entire engineering process. MDE approaches focus on representing systems aspects, such as behavioural or structural parts, through models and then translating them into code artefacts instead of directly programming the software component. As reported in [7,15,20,63,77], an MDE approach can provide support in modelling solutions for a single device development, as well as for more complex and fully functional IoT solutions [25,30,58,62,67]. To the best of our knowledge, there are not full-fledged solutions that provide customisation and reusability supporting mechanisms within a fully operational MDE approach. At the same time, the capability of representing the variability of the target IoT domain is in general not fully explored. In particular, almost

all the works we could find use domain-specific modelling languages or BPMN models, that are not able to provide a full representation of all IoT elements.

On the other hand, the use of feature models to address variability within the IoT context is suggested in [7]. In this case, their primary use relates to the handling of variability concerning the different functionality that a software product can include and the relation among them, providing reusability capabilities among different implementations [9,13,50,70]. In this sense, proposed approaches allow the definition of configurations that permit the selection of specific functionality among the ones included in the model, so to satisfy the overall formula subsumed by the feature model. The configuration is closely linked to the relationships that bind the various features within the model. Different configurations can be seen as different variants of the same product, as described in [85]. In [7,15], the authors highlight that the use of feature models presents positive evidence for the adaptation to different IoT contexts. Nonetheless, despite the various efforts, no single approach emerged as generally applicable to permit its adoption in all the possible IoT development contexts.

IoT Middlewares and platforms Internet of Things middlewares and platforms are considered helpful solutions that facilitate communication and data handling among elements that were not originally designed to be interconnected, and would not otherwise be capable of communicating. IoT middlewares can manage the communication between IoT devices, providing a more abstract level where communication interfaces between devices and IoT applications can be built up [33].

Among the most used IoT middleware, it is possible to find MiddleWhere [69], OpenIoT,[3], and FiWare.[4] They allow discovering and managing devices, providing analytic data services to manipulate and analyse upcoming data from devices, and sending valuable data to platforms/servers to build IoT applications. In such respect middlewares generally permit to address variability aspects connected to device communication.

IoT platforms have emerged to provide support to IoT applications development. They share the spirits with which middlewares were developed to interconnect different devices, and they offer functionalities able to perform operations over devices data, and to permit the entire IoT application development.

Currently, the main emergent option to develop IoT applications inside an IoT platform is to exploit the use of low-code development (LCD) environments [44]. Definition concerning the LCD can also be found under the umbrella of no-code development, and rapid application development. LCD aims to resolve the complexity of building IoT applications by reducing software application development time and decreasing the hand-coding required to develop the solution. It usually adopts a visual environment, which allows for building IoT applications more efficiently than traditional methodologies. In addition, widely used open-source IoT tools such as Node-RED,[5] Crosser[6] and NoFlo,[7] as well as IoT platforms such as IBM Cloud Platform,[8] ThingsBoard,[9] Losant,[10] and SmartWorks,[11] have incorporated this programming paradigm inside their solutions. To develop an IoT application according to the LCD paradigm, IoT platforms exploits the usage of a programming paradigm called flow-based programming (FBP), proposed in 1971 by Paul Morrison at IBM Canada [59]. The IoT application development based on these platforms and tools is usually provided through the use of graphical user interfaces that enable the definition of the applications through the interconnection of different graphical components [45]. Despite traditional software development, where a programmer has to write code to develop the desired solution, LCD hides all the classic programming concepts and focuses only on how the desired system should work, allowing drag-and-drop of prebuilt and reusable components to develop a system. In this sense, this type of development highly enhance reusability capabilities.

An example of LCD for an IoT application is reported in Fig. 1, where data are gathered from two temperature sensors components (read temperature sensor 1 and 2) and are directed to a computation node that computes the average value-producing as output. The complete application can be seen as a connection of different components, where each of them provides specific functionality and sends data to the next one. It is possible to facilitate the interconnection between heterogeneous devices using LCD to realise IoT applications suitable for small-medium domains (e.g. smart home, and smart building) [42,66,74], and large scenarios (e.g. smart city, smart logistic, smart hospitals and smart military environments) [47,48,75].

Discussion and limitations of the available solutions The categories of solutions proposed above only partially solve one or more of the identified challenges. On the other hand, they often bring other issues that can increase the complexity of the development of an IoT application.

[3] OpenIoT: https://www.openiot.in.

[4] FiWare: https://www.fiware.org.

[5] Node-RED: https://www.nodered.org.

[6] Crosser: https://www.crosser.io.

[7] NoFlo: https://www.noflojs.org

[8] IBM Cloud Platform: https://www.cloud.ibm.com.

[9] ThingsBoard: https://www.thingsboard.io.

[10] Losant: https://www.losant.com.

[11] SmartWorks: https://www.altair.com/smartworks.

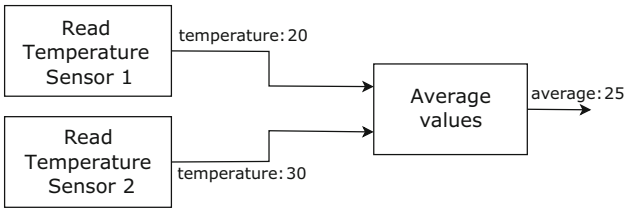


Fig. 1 An example of a low-code development IoT application

IoT middleware seems to be an optimal solution to use in handling the variability concerning IoT devices communication protocols. However, as described in [4] they bring several problems in relation to the rapid growth of the IoT domain and its continuous variation. On the other hand, IoT platforms exploit middleware's principle and provide clear support to decrease the complexity of IoT applications development, enabling the reusability of parts of an IoT application through LCD environments and elements. However, as depicted in [55], there is still not much support to build and reuse entire solutions across different application contexts, as those are generally considered stand-alone projects. Furthermore, allowing non-expert developers to produce code for IoT applications could lead to errors in the derivation of the application logic. Users often approach these environments without sufficient knowledge to build a correct application. In addition, many of these solutions, while on the one hand facilitate the development through the interconnection of graphical components, on the other, they require to insert specific technical IoT information. This information is usually technology-specific and differs for each IoT device. Overall, there is a general lack of tools and methods to support IoT application development.

A solution as it could be a *workflow management systems* can be applied in the IoT domain to model the IoT application behaviour. As mentioned in [19], proposing a business process oriented solution, it is clearly necessary to enrich the adopted modelling language with IoT-aware constructs to include in the business processes model aspects related to the IoT domain. In this sense, despite all the effort made by researchers, no modelling languages or notations affirmed itself for general usage in this context.

Indeed, [22] highlights a lack of standard and an extensive consequential fragmentation in proposing modelling notation to integrate IoT constructs inside business processes. In addition, this modelling notation enrichment with IoT related concepts must be coherent from the design to the process execution. Despite its significant usage from a modelling perspective, as highlighted in [41], support is also needed for the execution phase. To overcome these problems, it would be necessary to extend the used notation to include IoT domain elements, and coherently to extend the corresponding workflow engines to execute them. As described in [83], IoT-aware business processes are executed by process engines that are bounded to specific device types. The authors highlight how this decreases business process models reusability, as they need to be deployed in multiple or different IoT scenarios with different device types which provide similar functionality, that instead could be considered interchangeable from a customer perspective.

To deal with the problems mentioned above, developing and applying a *model-driven engineering* approach seems to be a possible solution for building IoT applications. They use models to decrease the overall complexity of building IoT solutions. In [12], the opportunities that the IoT domain could receive in adopting an MDE approach are highlighted. However, the authors also highlight open questions regarding the effectiveness of developing and building complex IoT applications to decrease the overall complexity and reduce time and costs. These applications must correctly handle the IoT domain variability in representing IoT elements and their relationships. In addition, aspects regarding the enhancement of the customisation and the reusability of IoT applications development were detected as relevant points to decrease the overall IoT development life-cycle time. We highlight how feature models seem to be an excellent candidate to model product lines of a complex IoT scenario. to represent and handle aspects related to device variability and the necessary functionalities to be supported by an IoT application. As their intrinsic property, they also provide reusability and customisation through different model configurations. However, a methodology that fully exploit these models and helps in building customised solutions seems not to have emerged, yet. More in general, enterprises and the research community still require a lot of effort to provide a common or standard MDE approach that can be practically used.

From the analysis we conducted emerged that no single solution, for the development of an IoT application, seems to address all the challenges mentioned in Sect. 2. On the other hand, a recently emerging trend, as described in [24,28,46], suggests the use of the model-driven engineering approach to develop IoT applications based on low-code development platforms. The authors highlight the potential benefits that could be achieved by combing Model-Driven development, possibly based on feature diagrams, and LCD. To the best of our knowledge, no concrete approach following the mentioned direction has been proposed, so far, being *FloWare* the first full-fledged proposal.

4 The FloWare approach

This section presents *FloWare*, a model-driven engineering approach conceived to support the modelling and development of IoT solutions. In the following, we provide an overview of the approach by describing the various *phases*

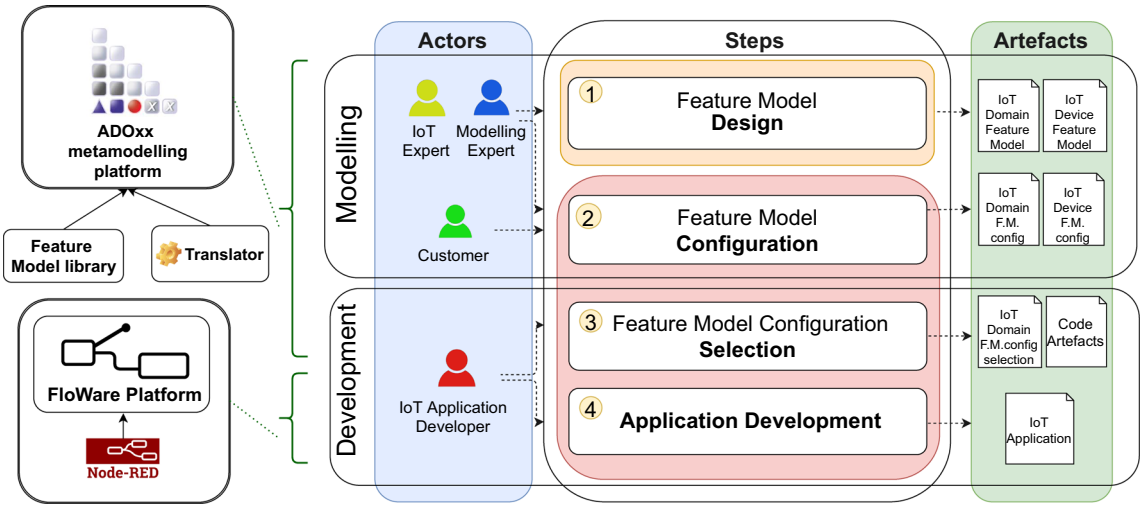


Fig. 2 Illustration of the FloWare approach with the supporting toolchain. (The area that divides step one from the others indicates that step one is performed only the first time the enterprise applies the approach)

and *actors*, as depicted in Fig. 2. Then, for each phase, we describe the foreseen *steps*, and the produced *artefacts*. In the following, we also describe the support we provide in developing and using the approach through a toolchain. The toolchain covers all the *FloWare* phases, from modelling to development phases. Additional information regarding the toolchain is reported in the FloWare web page.[12]

4.1 FloWare overview

Since the IoT development process is broad and concerns the intersection of multidisciplinary concepts by the various actors involved, converging towards a division of roles and concerns is certainly desirable. In our proposal, we intend to foster the *separation of concerns* principle [11], identifying and supporting different modelling, refinement and development steps. The approach aims at supporting from the modelling to the development of IoT applications, independently from the considered application domain.

Phases The approach is divided into two main phases: the *modelling* phase and the *development* phase. The modelling phase involves the design and usage of models representing the entire range of solutions that an enterprise can provide for a specific domain. Indeed, an enterprise specialising in a specific IoT domain generally offers many different IoT software and hardware solutions to the customers. In doing this, different customers come to it with particular requirements. Handling each requirement means for the enterprise that each solution is heavily customised based on specific necessities. Correctly producing customised IoT applications for each customer in reasonable time and costs are the main aim of the *FloWare* approach. Below, we refer to the term *crystallised knowledge* to indicate the possibility of representing the entire experience and awareness that a given enterprise acquired in a specific IoT domain.

Applying our approach, we provide a modelling structure to catalogue the entire enterprise knowledge for a specific IoT domain through feature models. We use feature models as a basis for representing and cataloguing hardware in the form of devices that could be installed inside an environment, and the system functionalities that the enterprise can provide to the customers. In this sense, our approach provides the possibility to handle the *complexity* in crystallising the knowledge regarding the variability of all these possible functionalities and the possible devices that could be needed to provide the defined functionalities. In this way, feature models help in managing the *variability* that derive from the target domain and the used devices, as described in Sect. 2. The feature model structure is built upon a well-known IoT ontology called IoT-Lite[13] to represent IoT devices, and systems characteristics, entirely.

Based on customer requirements, different feature model configurations can be defined over the same model. These requirements are derived from the customer needs and specifically referring to the specific application scenario (e.g. a building may need a smart access control system while another one may not). In this sense, we apply the *reusability* concept, providing the possibility of developing different configurations starting from the initial feature model, according to specific requirements. In this way, through these configurations, our approach aims to provide better *customisation* concerning all the possibilities that an enterprise can offer to the customers, so that they can benefit from this level

[12] FloWare web page: https://www.pros.unicam.it/floware.

[13] IoT-Lite: https://www.w3.org/Submission/iot-lite.

of customisation to better fulfil their requirements. At the same time, enterprises can offer customised solution packages that can be realised in reasonable time and costs.

The development phase supports the development of the final IoT application, and it starts from translating the feature model configurations into code artefacts. These artefacts store information regarding the IoT devices and systems involved in the final solution and are also used to provide a customised development environment. This environment is automatically filled with all the necessary information regarding the used devices. The approach also provides the developer with a selection of automatically generated IoT application templates, in the form of low-code development IoT applications, that the developer can use as a starting point to elaborate data coming from the IoT devices, visualise, and interact with them.

Actors The IoT application development is a multidisciplinary process that intersects heterogeneous knowledge from different involved actors [64]. In our approach, we revisited and adapted a classification of IoT actors and their roles as described in [52]. Our revision intends to better fit a model-driven approach as *FloWare* is. In the modelling phase, we require the involvement of 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 be deployed) within the application scenario. These enterprise actors collaborate with the *customer*, who requested the work, to define configurations of the IoT scenario that better fulfil needed requirements. The Development phase requires an *IoT application developer* to exploit the potential provided by the approach to develop IoT applications. In particular, the IoT application developer has the task of selecting systems and devices to derive a valid configuration as prescribed by the feature model, so to develop the IoT software application that will involve the selected systems and devices. In developing the IoT application, the developer will have to manage the interconnection of identified devices, and correspondingly of the produced data.

4.2 Modelling phase

The modelling phase involves two different steps. The first asks to an enterprise to model a complete IoT solution through feature models. This solution must incorporate all the functionalities and related devices that the enterprise can provide for a target domain. The second step asks to configure, following the customer necessities, the desired IoT application, and to choose which functionalities and devices to incorporate. To fully support these steps, we provide inside the ADOxx metamodelling platform a feature model library that can be used to design and configure feature models. Our library can cover all the steps, starting from the feature model design to its configuration, permitting then the generation of artefacts describing different aspects of the IoT application. Thanks to an intuitive graphical interface the resulting platform makes it easy to design and configure all the needed application details so to derive a fully functional IoT application.

In the following, we provide a detailed description of each step.

Step 1. Feature model design The enterprise that has decided to adopt the *FloWare* approach involves the ME and the IoTE actors to gather all the knowledge related to the reference IoT domain and related devices that it can offer. This collection forms the basis for designing a feature model for that specific domain. While the enterprise could design feature models using a custom terminology for describing systems and devices, we encourage following and applying the widely used IoT-Lite ontology as a reference [10]. This ontology constitutes a lightweight solution for the definition of concepts related to IoT devices and systems. The IoT-Lite ontology is used in different research works [73,76,78], to allow the easy development of IoT systems taking into consideration device heterogeneity and interoperability aspects among devices and systems. In our approach, we provide a feature model structure upon which it is possible to model one's own IoT domain. This structure is composed of two interrelated feature models, as shown in Fig. 3.

The first feature model, defined as *IoT domain feature model*, concerns the modelling of an entire IoT domain, in which, according to the interest of the IoT solution provider, all the systems and subsystems composing an IoT solution, can be inserted. For each of them, the relative devices need for the development phase, are reported. The second model, called *IoT device feature model*, is the one that allows the modelling of a device with its characteristics and properties. It can be used as a ready-to-use model to express device features, or possibly extended to add emerging future characteristics. Our approach permits the inclusion of this model to cover all the aspects reported in the IoT-Lite ontology.

In the following, we describe the IoT domain feature model referring to a smart campus IoT domain example, and the IoT device feature model, as a model representing a generic device. These models play the role of platform-independent models (PIMs), which allow the enterprise to represent all the systems it can provide to various customers, independently from any technical specification. These models do not contain any specific information about the target scenario; instead, they provide a high-level abstract representation of the entire reference domain.

IoT domain feature model After analysing all the IoT systems and devices that the enterprise can provide for the smart cam-

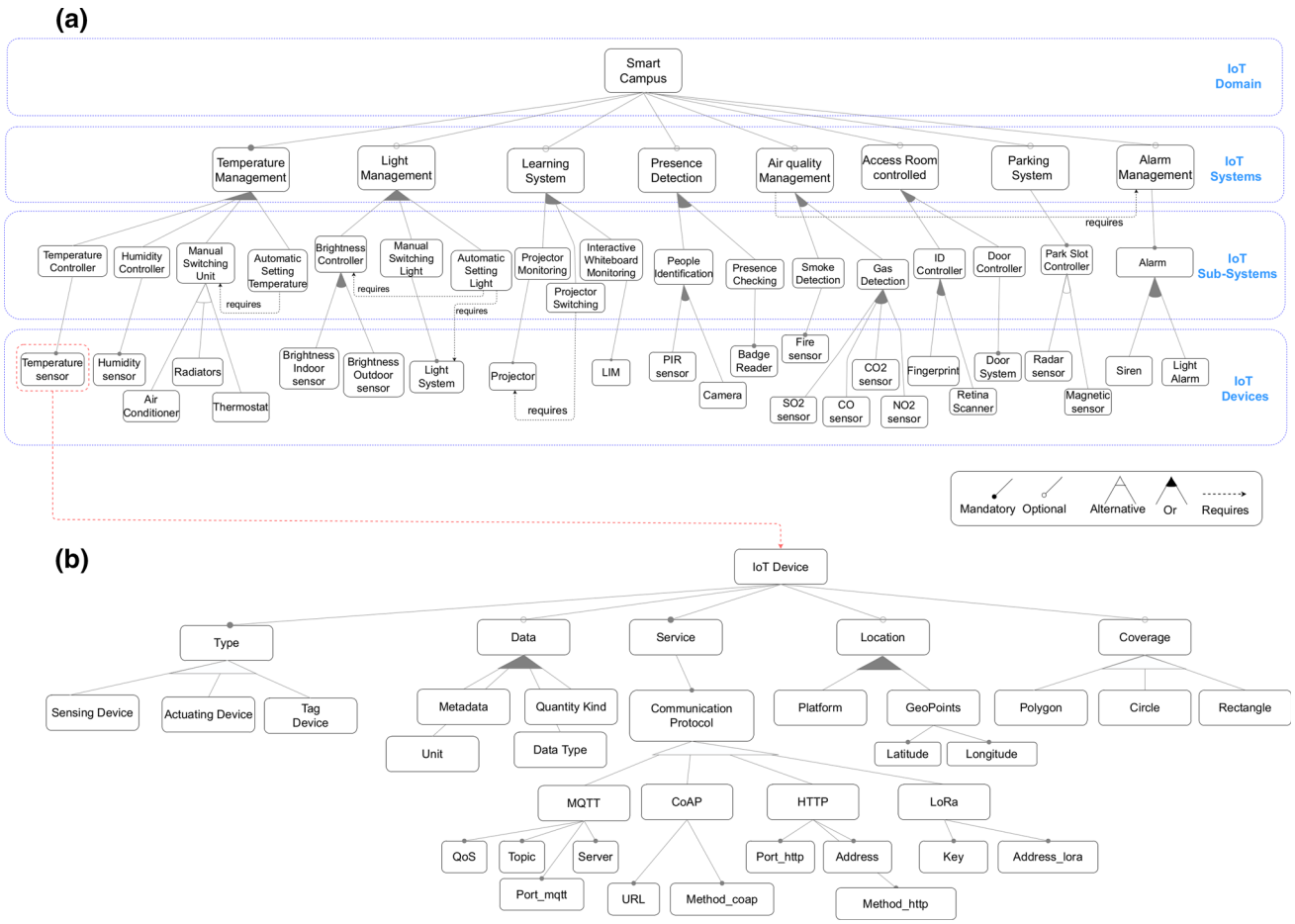


Fig. 3 Feature models of the smart campus (**a**) and of the IoT device (**b**)

pus scenario, the ME and the IoTE can design the IoT domain feature model. By using the IoT-Lite terminology, this model synthesises all the knowledge acquired concerning enterprise provided solutions.

The ontology describes an IoT domain as a collection of systems, which in turn can be decomposed into sub-systems. Different IoT devices can be associated with each sub-system. According to several works [2,61] that summarise the functionalities that a smart campus can provide, in Fig. 3a is reported the resulting feature model that the enterprise could have derived.

The provided feature model structure is the following. At the first level, we reported the reference **IoT domain**, in this case, the smart campus. At the second level, it is possible to represent all the **IoT systems** involved in that domain (e.g. temperature monitoring system, a light management system, etc.). The enterprise experts are in charge of including, defining and characterising these systems.

As the IoT domain is very large, the systems in this scenario are also heterogeneous. As described in the model of Fig. 3a, many IoT systems that are part of the domain can be decomposed and detailed through one or more sub-systems. For instance, the temperature management system could be a complex IoT system that involves many different IoT sub-systems, to give the possibility to monitor the temperature, and to manually or automatically adjust it. At the bottom level of the model, the **IoT devices** that contribute to support the IoT systems and sub-system operations, are reported. While designing a feature model, it is possible to define constraints among the various features, specifying whether a configuration has to abide by the following relations: *mandatory*, *optional*, *alternative* or *exclusive*. In addition, it is possible to express external constraints that allow the *require* or *exclusion* of a specific feature from a configuration. An example of a constraint could be the relation that binds an air quality management system with an alarm system. In case this relation is present between these systems, the latter must be included in case the first is included in a configuration. The type of relationships among the various features of the model is provided by the enterprise, which, being expert in a given domain, provides a complete overview of all its possible services.

IoT device feature model The second feature model that the enterprise can define is used to collect all the knowledge about the types of IoT devices that the enterprise manages. An example of an IoT device feature model is reported in Fig. 3b. We defined this model following the device characteristics highlighted in the IoT-Lite ontology. However, it is possible for an enterprise to extend and modify this structure to better fulfil their needs.

The presented IoT device feature model is characterised by five main features: type, service, data, location, and coverage. **Type** is a mandatory feature to express the IoT device typology. An IoT device can be a *sensing device*, if it provides information from the physical environment (e.g. the temperature of a room), an *actuating device*, if it can act on the environment (e.g. a device that can open a door), a *tag device* such as RFID, NFC or QR-codes. **Service** is a mandatory feature that represents the way IoT device related information can be sent/retrieved. Usually, the main *communication protocols* used to send/retrieve information within the IoT domain are HTTP, CoAP, LoRa or MQTT. **Data** is an optional feature that groups all the information regarding the data sent/received to/from an IoT device. It is considered an optional feature as in case it is missing this does not affect device operations. These data can be catalogued into *metadata*, any metadata that a sensor can provide, *QuantityKind* and *unit*, which are abstract concepts to represent what can be measured and the measurement unit (e.g. temperature as quantity and Celsius as a unit). Other information necessary to allow the correct manipulation of the device data refers to the *DataType* that the device can handle. Regarding the data that the device sends or receives, it is possible to define if they are numeric values, Boolean, etc., to allow the system to correctly process them. **Location** is an optional feature representing all the information regarding the physical IoT device position, including the latitude and longitude in *GeoPoints* and the physical *platform* where the device is located (e.g. desk number one). **Coverage** is an optional feature that refers to the specific coverage range that the device provides (e.g. a temperature sensor inside a room has a coverage of the entire room). Coverage can have different forms such as that of a *polygon*, a *circle*, or a *rectangle*.

Once Step 1 of the approach has been performed, and the model representing the entire IoT domain has been designed, it will not be needed to repeat it for each new customer. Instead, all the other steps must be repeated every time a new customer requests an IoT smart campus solution. In this sense, it is possible to provide *reusability* mechanisms starting from the IoT domain feature model designed by the enterprise, and its relative IoT device feature model. In Fig. 2, the separation of Step 1 from the others highlights this concept. The established feature models will provide a base for defining the various configurations requested by the different customers. In addition, we provide a way to categorise and handle the *variability*, giving the possibility to represent all the possible characteristics that the IoT domain, as well as IoT devices, can have. It is possible to extend the already designed models by adapting them to meet advanced requirements that the enterprise can adopt, providing new additional systems or modifying those already present (e.g. removing those that are not offered anymore by the enterprise).

Step 2. Feature model configuration After the enterprise designs its range of possible solutions for a target domain, it can put *FloWare* into practice to answer the needs of a customer. The enterprise experts and the customer use the IoT domain Feature Model to discuss and configure all the necessary solutions, following the specifications reported in the model. Considering the the smart campus scenario, a customer may need to build IoT applications for different departments of the same university. Each of them have its peculiarities and diversities. Different configurations representing departments from an abstract perspective can be generated. At this stage, no specific information are requested in these configurations. An example concerning a department configuration is reported in Fig. 4a, showing all the mandatory systems that the department must have, such as the *temperature management* and the optional systems that can be chosen, such as *light management* and *air quality management*. In addition, the *Alarm* system is included in the configuration, given the rules that bind the air quality and the alarm systems. All the sub-systems that the department needs are also selected for each selected system. For example, the temperature management system reports different variability points in the initial feature model, and various sub-systems can be chosen when configuring it. In this configuration, the sub-systems that allow both a manual and an automatic control of the department temperature have been chosen.

Successively, the necessary devices have to be selected. It is possible to notice these variability elements in the selection of devices. The entire temperature management system could be realised in one location using a combination of temperature sensors and an air conditioning system, while in another using the temperature and humidity sensors and the radiators. This variability allows producing different configurations starting from the same model and taking into account the specific needs and intentions of a customer. As shown in Fig. 4b, the example refers to two different devices included in the configuration: the temperature sensor and the air conditioner. These devices differ in their configuration. As for the temperature sensor, it is possible to notice that it has been configured as a sensing device, as it is used to capture the room's temperature values. The location points of the device, intended as latitude and longitude GeoPoints, have been selected in the configuration. The service that allows to read/send data is also configured. In this case, the MQTT protocol was selected. Finally, the specification of the type

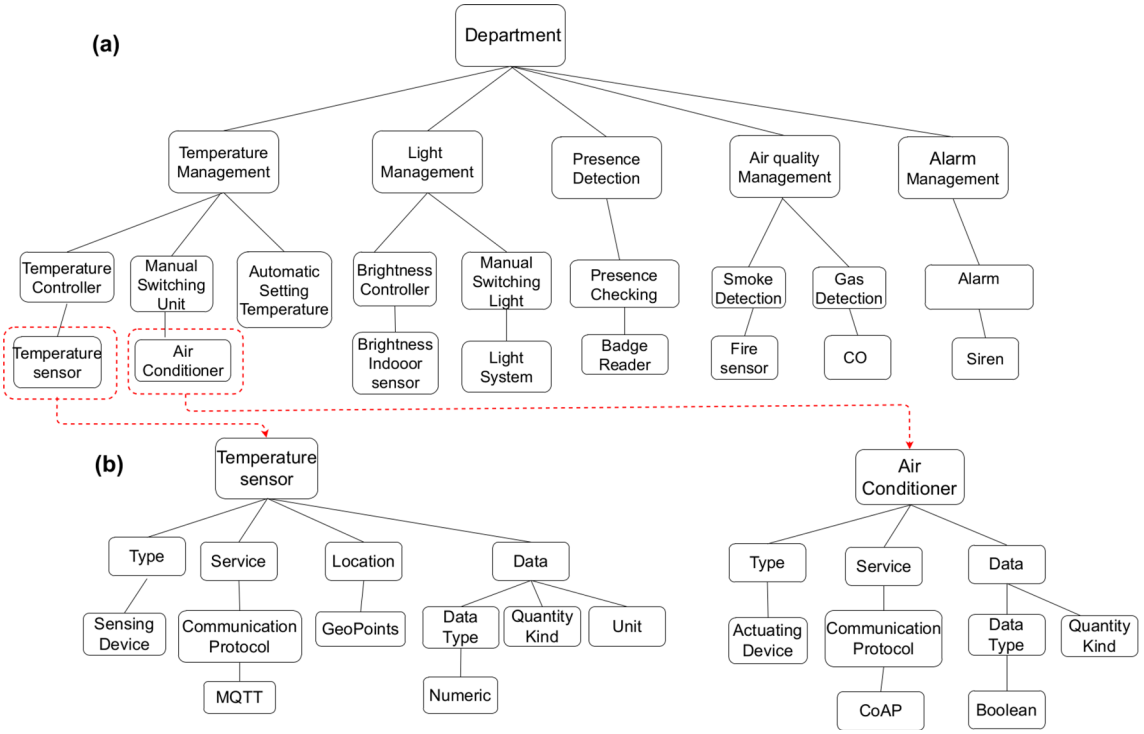


Fig. 4 Example of **a** a department configuration; **b** the temperature sensor and air conditioner devices configurations

of data sent has also been selected in the configuration. In this case, the numeric type has been configured under data type. Additional information such as the quantity kind, and unit is also inserted.

Referring to the air conditioner, the configuration slightly differs, and being a mandatory selection the type of device has been specified in this case. In particular, it has been configured as an actuator as this device performs the function of changing environmental parameters such as the temperature value. The type of communication protocol is also different. In this case, CoAP has been selected. Finally, the data parameter that the device provides is configured as a Boolean data type, as the device provides information about its status (e.g. whether it is on or off), and information regarding its quantity is also selected.

At this point, all the features chosen in the configuration include the devices that need to be configured by the ME and the IoTE actors. After configuring a department, it is necessary to provide technological specifications regarding all the included devices. Such information can come either from the customer, that could have already deployed such devices to support other IoT application scenario, or directly from the IoT solution provider. This information is dependent on the device's technologies, and their specification will result in so-called platform-specific models. For example, specific information based on the type of communication protocol chosen (e.g. if it is MQTT, the broker server name, the topic, the listening port, etc.) or the exact latitude and longitude of GeoPoints are requested. All the specific configurations are saved in a common repository to have the possibility to reuse them as necessary to develop the complete IoT application. Indeed, the IoT application developer will use them to select which IoT systems and devices will be included in the final IoT application.

4.3 Development phase

All the defined configurations with specified devices information will then be made available to the IoT application developer, i.e. the expert in charge of developing the IoT software application to be deployed.

Step 3. Feature model configuration selection The IoT application developer can select one configuration related to a specific model to develop the application or choose multiple configurations so to derive a more complex application.

Taking as a reference the configuration reported in Fig. 4a, which is provided with a representation of all the IoT systems for the department, the IoT application developer can select for which systems and sub-systems to develop the IoT application.

As an example, the IoT application developer decides to develop an IoT application for the temperature management system. After the IoT application developer select which IoT systems and devices to use to develop the application, the predefined transformation will permit to derive a set of snip-

```
<configuration>
//device name
<feature automatic="selected" name="Temperature sensor"/>
//service specification
<feature automatic="selected" name="Service"/>
<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"/>
//data specification
<feature automatic="undefined" name="Data"/>
<feature automatic="undefined" name="Data Type"/>
<feature automatic="undefined" name="Numeric"/>
</configuration>
```

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

Fig. 5 A detail of a Temperature sensor device XML configuration translated to a JSON component, readable from our FloWare platform

pets of code, defined as *code artefacts*, specifically conceived and formatted according to the requirements of the selected platform to be used. These code artefacts are built through a *translator* script that translates the device's information into the format processable by the target IoT platform.

In particular, the translator takes the information regarding the selected systems (e.g. the temperature management system) and the selected devices (e.g. the temperature sensor device) and translates them from XML configurations into code artefacts (e.g. JSON file) that the target IoT platform can process.

An example showing this translation is reported in Fig. 5. The figure shows a part of an XML file that corresponds to a feature model configuration of a device reporting the selected features and the specified device data. These data are the name of the sensor (temperature sensor) as well as some details concerning the service by which device data are accessible. In particular the protocol (MQTT), the server name (https://www.brokerserver.com), the server port (1883), the quality of service for the protocol (0 means there is no guarantee of delivery), and the topic at which to subscribe for accessing the device data (room1/temperature). In addition, specific information regarding the data type that the device exposes is reported (numeric type). It is worth noticing that, referring to the IoT device feature model in Fig. 3b, all the mandatory features have been included and are those with the automatic attribute set to "selected" in the XML configuration file. The selected features that are optional inside the model present the automatic attribute set to "undefined".

In our approach, we provide some script to translate the XML configurations into readable code for our FloWare Platform. We integrated Node-RED, one of the most used low-code development environments for building and running IoT applications inside the platform. The example in Fig. 5 shows that this configuration information is translated into the readable Node-RED format, that is, a JSON format, ensuring the correct correspondence between the configuration and the derived translation. The provided translator also elaborates the information collected inside the XML configuration to generate basic application logic. These basic templates allow retrieval of the information from the devices to perform simple operations inside the FloWare Platform. We defined those templates based on our experience. However, it is reasonable to assume that any enterprise or user of the *FloWare* approach can define more complex templates based on the gathered experience in developing IoT applications.

In *FloWare*, templates that provide basic application logic to interact with the selected devices are automatically generated to support the developer in designing more complex applications. An example of a provided application logic template for our scenario is the one related to the interactions with the temperature and air conditioner devices, reported in Fig. 6a. The templates appear to the developer as interconnected *nodes*. This interconnection of nodes corresponds to the basic application templates that we provide to support the IoT application developer. This basic application logic is enough to handle the automatic import, elaboration, and visualisation of the device's data. The low-code development environment transforms the application logic that is graphically displayed into executable code (usually using JavaScript or Java languages), which allows its execution. In this way, the user of this environment is free from having to write code by hand. Developing the application through interconnectable and reusable graphic components is only necessary.

For what concerns the temperature sensor, as its data type is numeric, as defined in the configuration step, nodes that

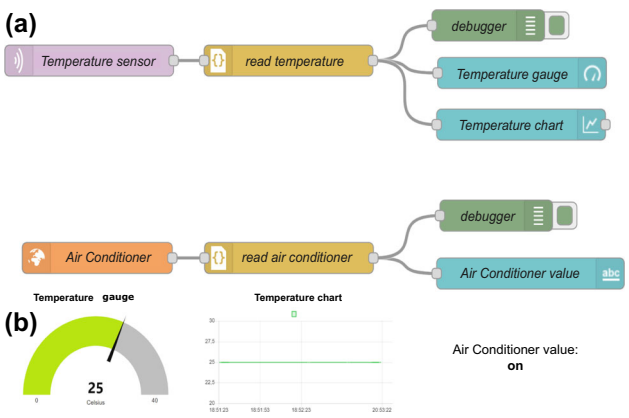


Fig. 6 Templates automatically generated for the Temperature sensor and an air conditioner devices (**a**) and the obtained dashboard (**b**)

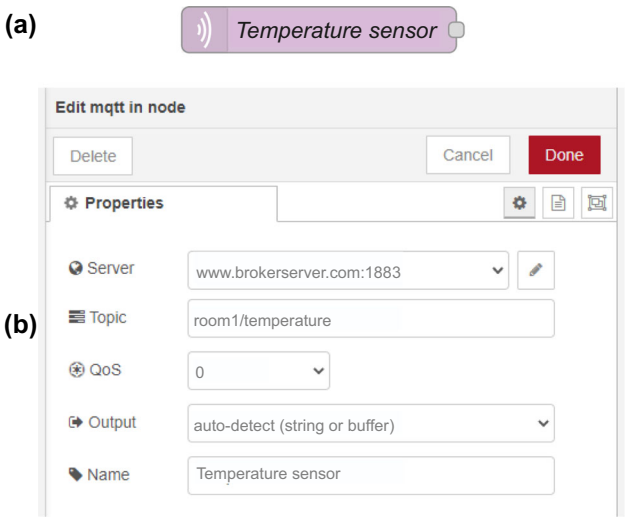


Fig. 7 Representation of a generated temperature sensor node (**a**) and of its related information (**b**) inside Node-RED

can correctly handle this type of data are provided (gauge and chart nodes). In contrast, as the air conditioner provides a Boolean value (true/false corresponding to on/off), its value is displayed in a textual format. In addition, in Fig. 6b the relative generated dashboard is presented. The dashboard includes different widgets that allow the easy visualisation of the temperature sensor and air conditioner device data in real-time.

Each node included in the templates is automatically filled with device information provided during the feature model configuration phase. In Fig. 7a, an MQTT node is generated and its specific information (Fig. 7b) are automatically retrieved from the configuration translation. The developer does not have to insert specific information regarding each device, but the environment automatically retrieves them from the previous configurations.

The definition and the usage of automatically built templates permits the *sharing of knowledge* between the various experts. Our approach provides such an abstraction level that permits the developers to remain unaware of the specific information regarding the devices, as the imported configurations automatically insert the necessary information. In addition, through this approach, it is possible to foster *artefacts reusability*, providing a starting point to generate the IoT application instead of redefining it from scratch.

Step 4. Application development In this step, the IoT application developer can adequately handle the code artefacts generated during Step 3 and provide them as input to the integrated platform used to define the IoT application logic. The generated code artefacts can be automatically imported into the FloWare Platform through a script that, from the configurations made inside the ADOxx environment, sends the code artefacts directly to the FloWare platform. The IoT application developer can then use the incorporated Node-RED environment to modify or directly execute them.

In this final step, the derivation of the whole IoT application takes place. To fully support the IoT application developer inside the FloWare Platform, we provide a script to automatically set the Node-RED environment based on the configurations. Indeed, the Node-RED tool presents a palette formed by many nodes that can be connected to design an IoT application. We automatically configure the Node-RED palette in such a way to display not only the default nodes but also those nodes that are needed to interact with the systems and the devices selected from the feature model configuration (Step 3 of the approach). Without this support, each node that does not come with the default palette would require a manual installation. The result of the configured environment is reported in Fig. 8a, where different nodes that are not provided as default are added (e.g. that of the highlighted CoAP nodes) and correctly configured. In this way, all the devices and information are automatically and correctly imported.

At this point, the IoT application developer has the complete environment completely set. He/she can retrieve all the information related to the included devices, and the basic templates corresponding to the available configuration, to possibly proceed to further manipulations and node interconnections. In particular is only necessary to use additional operational nodes to enrich the IoT application logic. An example of the extended IoT application is shown in Fig. 8b, where a more complex logic to manage the temperature in a room is provided automatically. Our approach has already generated a considerable part of the final IoT application. Indeed, the automatically generated nodes have been grouped within the green perimeter, so to see the valuable support provided by *FloWare* in developing the final IoT solution.

In our approach, we stress the importance of developing IoT software applications through low-code development environments, as suggested in [44,71]. The described process and composing activities illustrates how using *FloWare* it is possible to crystallise the knowledge of a specific IoT application context, to make possible its reuse in the development of several related concrete IoT applications.

5 Demonstration

In this section, we provide a practical demonstration of the *FloWare* approach in a smart campus domain to answer the second research question: "(RQ2)—How can we improve support for customisation and reusability of IoT application modelling and development?".

We introduce the smart campus application scenario, and we describe the motivations of the two main actors involved: an enterprise that wants to reduce the complexity of developing customised IoT solutions, and a customer in need of

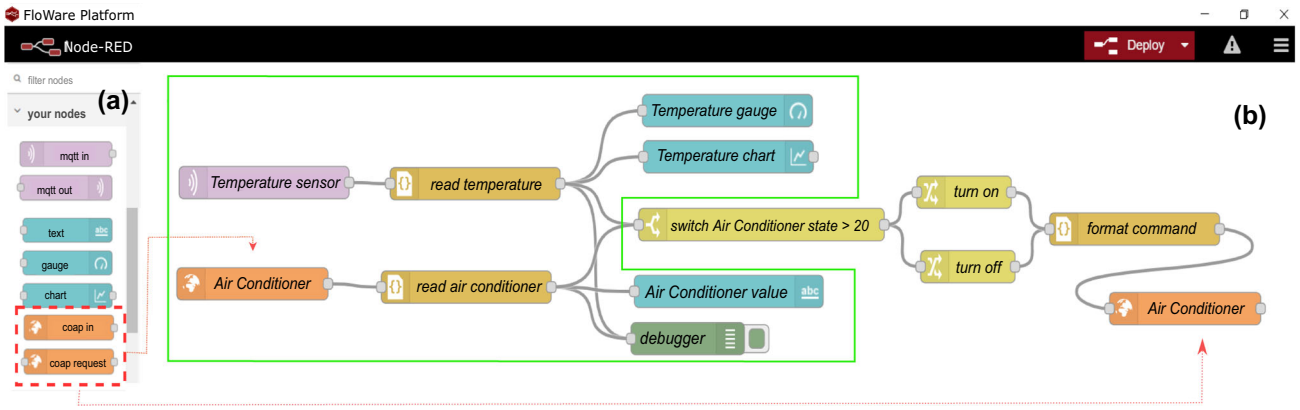


Fig. 8 The customised Node-RED environment (**a**) and a complete IoT application to automatically manage the temperature of a room (**b**). The application is developed by the IoT application developer combining and extending the templates of Fig. 6

an IoT solution to transform a traditional university campus into a smart campus.

5.1 Application scenario

A smart campus can be defined as a university campus equipped with IoT devices and exploiting their potential, becoming able to detect and to automatically apply changes in the environment, and its inhabitants' state [29,87]. Transforming a traditional campus into a smart campus can enhance education and teaching, and it can optimise life for students, employees, and visitors. It can also improve the quality and performance of the services, reduce costs and resource consumption, and engage more effectively and actively with its residents [16,54].

One or more managers can handle all the functionalities deployed within a smart campus through software solutions, generally in the form of graphical dashboards explicitly created to meet the needs of the smart campus. Using these dashboards, it is possible to monitor the smart campus in real time, to remotely manage devices activation (e.g. detecting the presence or the absence of personnel, it is possible to turn on/off the lights remotely via a virtual button), and to analyse historical data to plan future actions.

The IoT solutions provider (enterprise) The company we refer to has gained significant experience in digitising university campuses, providing ad hoc solutions for every situation encountered. Each solution is usually developed from scratch, increasing production times and construction costs. Referring to the same IoT domain (e.g. smart campus), IoT software development for different clients however clearly has shown some commonalities [79]. The company found that customers often required the same features (e.g. temperature management, light management, smart HVAC, etc.). However, each run project has often rewritten the software to handle those same features. In addition, customers can request the use of different sensors to create a specific functionality according to their needs. For example, cameras or presence sensors may be needed to monitor presence. The numerous dimensions of variability that we can observe in this domain are the primary source of problems for the software development company. This differentiation must be managed in a simple way for the company. However the management of each project as a single instance makes difficult to introduce improvements in all the IoT applications belonging to the same typology. All this leads to a high degree of complexity for the company in maintaining and reusing the acquired knowledge by applying the best practices achieved over the years.

The university In our scenario, the university plays the role of the customer as it intends to make its campuses smarter. The university decides to rely on an IoT enterprise's experience and start the necessary interactions to fulfil its interests. Given that a university can be seen as the combination of various buildings (e.g. departments, administrative offices, etc.) and the people that populate them (e.g. students, professors, administrative, etc.). Therefore the requirements for IoT solutions may vary based on the building where devices will have to be deployed, and the objectives of the people consuming the services.

Some practical examples can help to better clarify the scenario. Therefore we consider a generic university organised over various departments, and their corresponding and peculiar needs. So in our hypothetical university we have:

– the *Chemistry department*, in which, for safety reasons, it is necessary to have an intelligent system for air quality monitoring. This system can immediately report the air quality levels that may be influenced by the experiments conducted in the labs. In particular, it is necessary to include fire, CO, NO2 and CO2 sensors. In addition, to maintain the various chemical elements accurately,

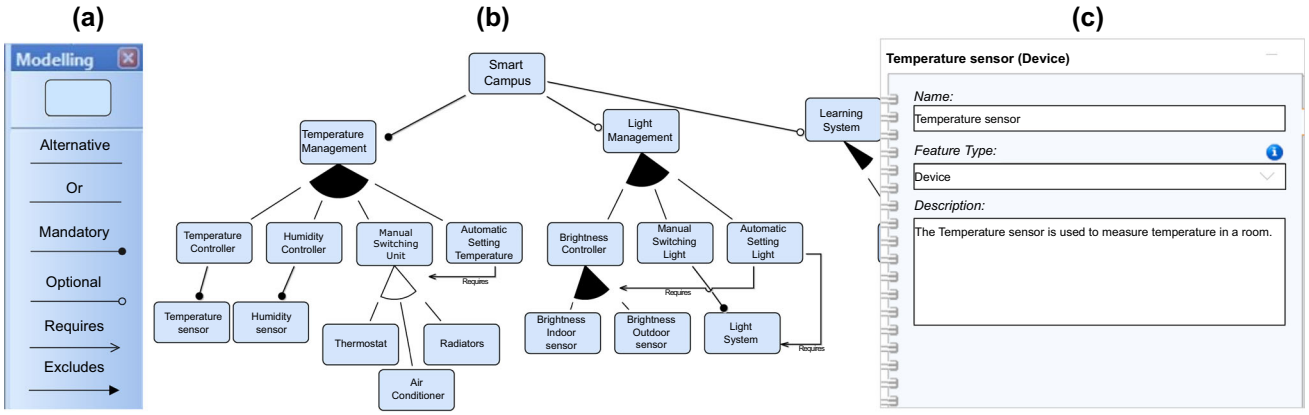


Fig. 9 A representation of the feature model library within ADOxx (**a**), a designed smart campus platform-independent model (**b**), and a detail on the *temperature sensor* feature (**c**)

the temperature should always be automatically kept between 18 and 21 °C.

– the *Biology department*, in which there is the necessity to monitor experiments involving plants. In particular, there is the necessity to have a system able to control and manually act on the entire temperature and humidity levels inside the labs so to manage in an optimal manner the radiators and the lighting levels for plants wellness.

After the university provides the requirements for each department to the enterprise, this has to develop a customised smart campus solution. In the following section, we report how the *FloWare* approach can help the enterprise to meet the university demands to provide customised IoT applications.

5.2 Smart campus case study

The company wishing to apply the *FloWare* approach can use the provided toolchain to design its functionality model. An example of the use of the provided toolchain is shown in Fig. 9, where the same smart campus scenario (shown in Fig. 3a) is modelled to show the feasibility to represent all the functionalities that can be developed in a complex scenario. In particular, we show how it is possible to design models through the graphics library provided (Fig. 9a). A partial representation of the entire feature model is shown in Fig. 9b. The experts in developing these models can then provide details on each feature inserted. These details can represent devices and systems involved in the scenario, as shown in Fig. 9c. In particular, in the example, the experts select the temperature sensor. Then, it is possible to enter information such as the name, a generic description of the device involved, and the type of feature it represents (in this case, a Device Type). It is worth noting that for the IoT solution provider, it is necessary to design this model only the first time it applies *FloWare* in such a way to represent the entire acquired knowledge. Indeed, our approach is strongly based on the possibility to *reuse* the initial feature model to provide all the necessary configurations to better explain and model all the requirements. At the same time it is possible to easily extend such a model to include additional functionality to be developed in a given context.

After enterprise experts have modelled the domain of interest, including all the functionalities it can provide to the customers and its related devices, it is possible to use this model to guide each customer requiring smart campus IoT applications. In this case, the enterprise collaborates with the university to fulfil its requirements. The university approaches the enterprise with several requests on how to develop the IoT solution to manage the entire smart campus, as described in Sect. 5.1. As explained, each department has its specific necessities and needs different devices and functionalities. In this sense, by applying the *FloWare* approach, the enterprise can ensure that the university can *customise* their preferences through different configurations of the designed feature model. The developed feature model is, indeed, *reused* to derive several department configurations. Experts in cooperation with the university personnel must select the necessary characteristics guided by the relationships defined in the model itself. In Figs. 10 and 11, it is possible to notice how different configurations can be produced from the initial model. In this case, the configurations for the *chemistry department* and the *biology department* are provided following the already mentioned requirements.

Once the configurations are defined and all the university requirements are satisfied, the experts have to provide the specific configuration for each device part of the selected systems. Correct devices configurations is the one reported in Figs. 10b and 11b. For the temperature sensor and the Humidity sensor devices, it is necessary to fill in a form to enter specific information about them. The elements requested in the insertion and its relations (e.g. mandatory, optional, and

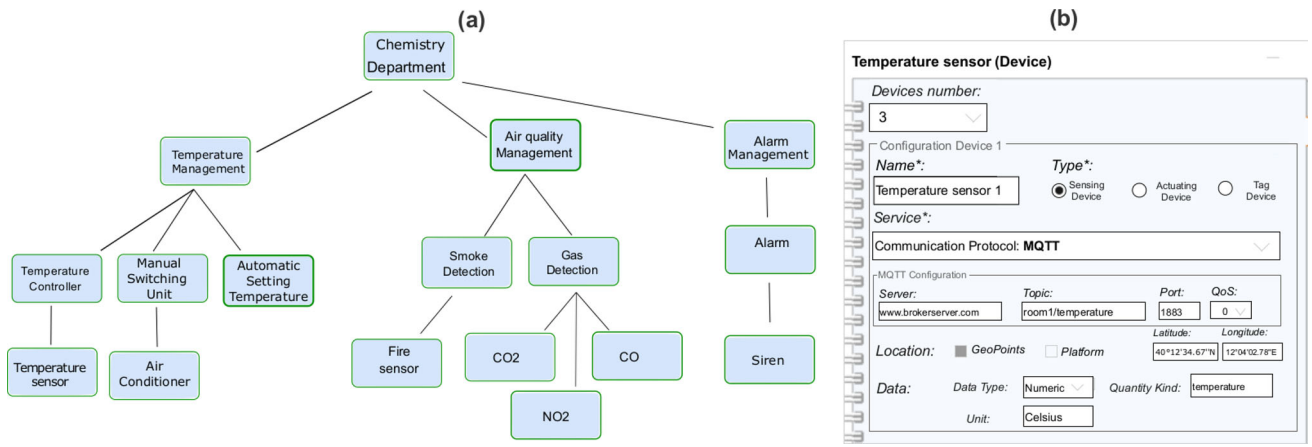


Fig. 10 A representation of the chemistry department configuration on ADOxx (**a**), and the temperature sensor device configuration in detail (**b**)

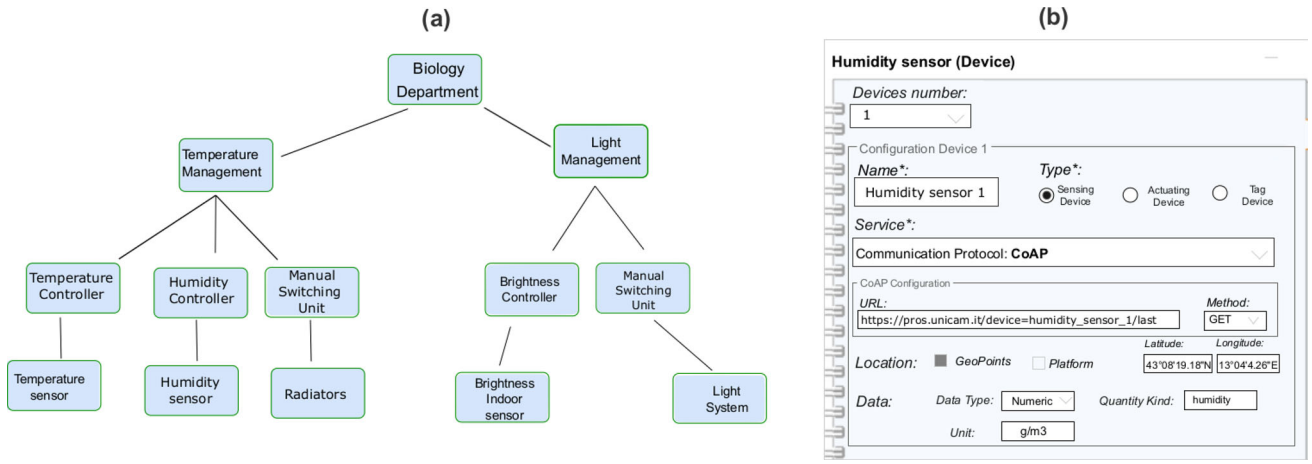


Fig. 11 A representation of the biology department configuration on ADOxx (**a**), and the humidity sensor device configuration in detail (**b**). (Asterisks in the device configuration indicate mandatory features that must be filled, following the IoT device feature model of Fig. 3b)

others) were already defined in the IoT device feature model in Fig. 3b and are represented in our library in textual form. For each device included in the configuration, specific information regarding the selection provided must be inserted into the form.

In the first case, three temperature devices are inserted, while the biology department only needs one humidity sensor. All of these devices need to be configured appropriately.

Despite the device name and its type, one of the main information required is the communication protocol provided to communicate and its relative information. In the configurations, different communication protocols are provided, which require different parameters. After a valid configuration is completed, both systems and devices defined are stored as an XML file in a dedicated repository.

The enterprise has stored all the configurations that reflect the university requirements and a complete definition of each device's information. Then, enterprise experts select on which target platform the solution has to be deployed in. In this sense, we provide the possibility to derive the entire solution for our FloWare platform, which already includes Node-RED, a low-code development environment able to run the developed IoT application directly. In particular, the IoT application developer chooses the interesting configurations and retrieves the automatically generated code artefacts. That codes are directly imported inside the FloWare Platform and provide the experts with an already configured environment and a ready to use IoT application in the form of Node-RED flows (as described in Fig. 7). At this stage, the IoT application developer has a complete customised environment according to the information inserted in the configurations and can extend the application logic provided by adding a more complex one. In particular, the IoT Application Developer has both the generated application for the two departments and can then extend those by adding additional nodes.

The result for the complete IoT application regarding the chemistry department is reported in Fig. 12a, where the devel-

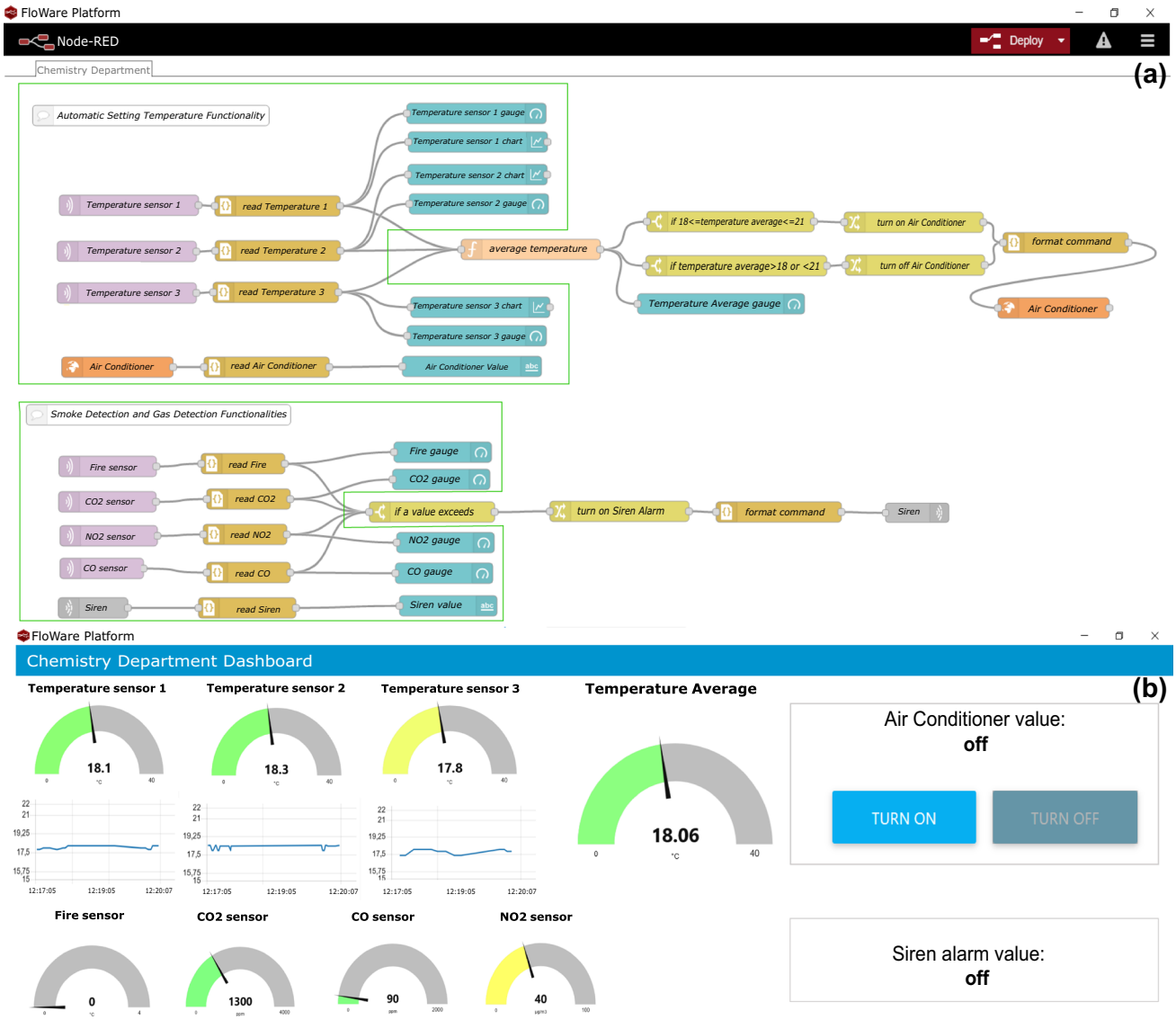


Fig. 12 The FloWare platform with **a** the complete IoT application for the *chemistry department*, **b** its related dashboard

oper extended the generated application by adding some logical nodes that allow setting the temperature to 20 °C. In addition, this extension includes the air quality functionality to detect if smoke or dangerous gases are present in the laboratory. If a problem occurs, the system automatically activates the Siren alarm. The outcome of this IoT application is a dashboard that allows visualising all the device's data collected in real-time, as shown in Fig. 12b. The nodes automatically generate almost all the dashboard widgets in the IoT application templates (these dashboard nodes are grouped inside the green area). To provide the possibility to show more valuable data, such as the department's average temperature and manually turn on/off the air conditioner, the IoT application developer extends the IoT application by adding additional nodes. This is reflected in the dashboard provided to the customer, which will contain essential information for the use and management of that department.

Similarly, the IoT application for the biology department needs to be developed. In this case, different requirements will result at first in a different configuration, and then in different templates, as shown in Fig. 13a. Following the department requirements, the temperature and light management systems are derived. Notably the choices made by the different departments lead to a variability associated to the different functionalities; at the same time, a functionality to be developed in two different configurations can also present a high variability. Indeed, the only commonality between the two departments is the chosen temperature management system. However, in this department, different features are required. In particular, the temperature is not automatically

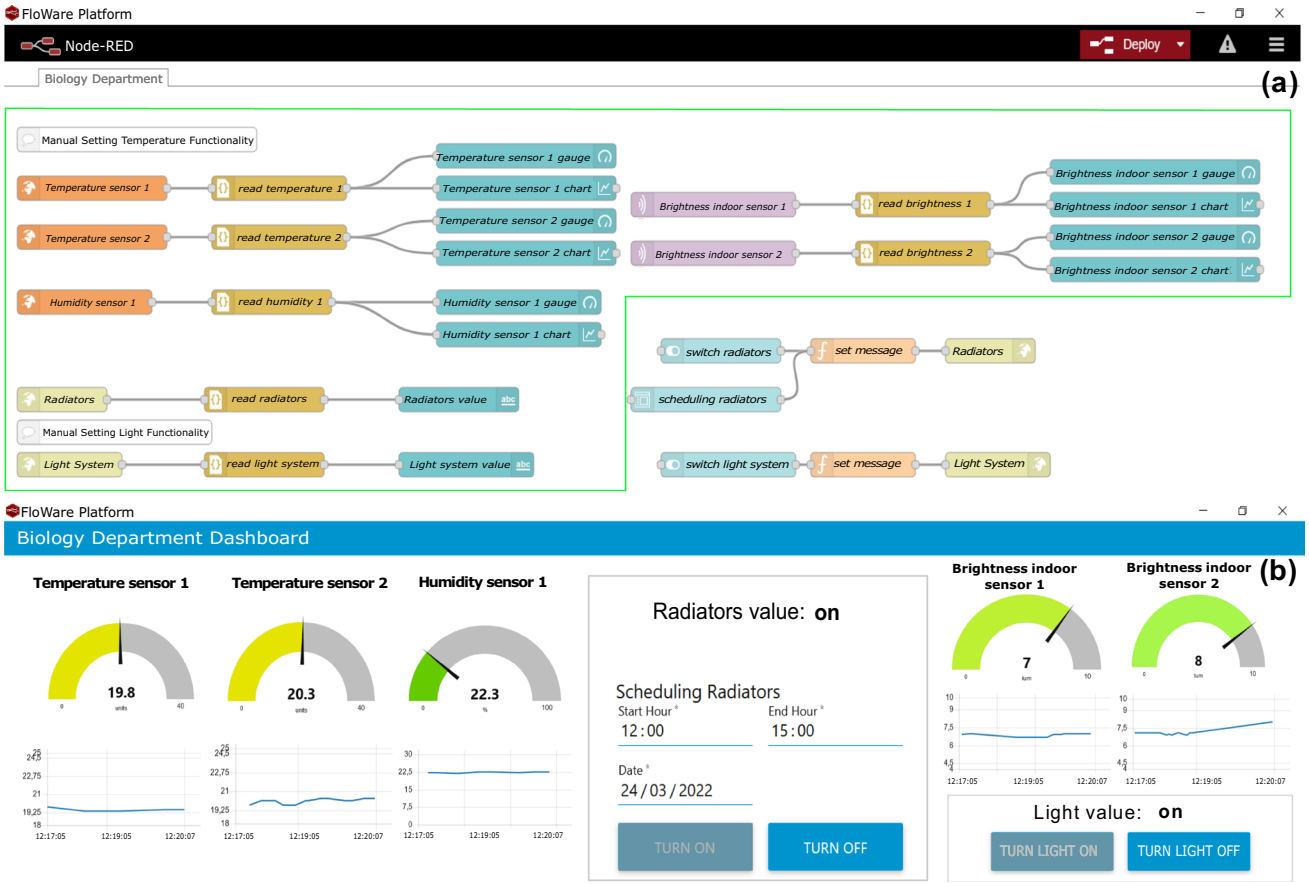


Fig. 13 The FloWare platform with **a** the complete IoT application for the *biology department*, **b** its related dashboard. (The part highlighted in green shows the automatically generated flows by the *FloWare* approach. The IoT application developer manually provides the additional part.)

set inside this department, and a system is necessary to interact remotely with the radiators to set them. In addition, the possibility to manually schedule the time at which to turn on/off the radiators for that department is provided. This department has also to be equipped with the functionality to remotely control the light system. These different configurations translate into different needs for the development logic of the entire application, and the devices to be used. This highlights the high degree of variability present in the whole IoT domain, both for the development of different functionalities, and also in developing the same one. After the IoT application developer extends the code artefacts provided with more complex logic, these changes are also reflected in the dashboard, in Fig. 13b. In this case, it is offered to the customer the possibility to visualise all the real-time data concerning the temperature, humidity, and light inside the department and manually change them through radiators and electric systems. In addition, it provides the possibility of scheduling the power on or off for the radiators.

Once all the department's IoT applications are derived, with corresponding dashboards, the final IoT application is completed. At this point, it can be delivered to the university, that then will be able to view all the data relating to its departments and structures, to apply decisions based on them, and to remotely interact with the university environment through the deployed devices. It is reasonable that the enterprise, to maximise the reusability factor, will store those developed IoT applications inside the FloWare platform and reuse them to provide other solutions.

Going back to RQ2, we illustrated in this section a possible approach to effectively support reuse and customisation of models representing generic IoT solutions. Therefore, we illustrated how an approach mixing feature models and LCD can concretely support the development of reusable artefacts for the derivation of highly customisable IoT solutions so to reduce the time and costs needed to derive a full-fledged IoT application. It can be certainly observed that the most challenging part is the creation of a high quality Feature Model that as said reports the knowledge that an organisation acquired operating in a specific IoT domain. Nevertheless once such knowledge has been correctly represented the derivation of different, but somehow similar, applications seems to be rather easy and relatively supported by the tool. Clearly, the one proposed by *FloWare* is only a possible pro-

posal, and other ways to reach similar objectives are certainly possible. Indeed, Sect. 6 intends to put in relation *FloWare* with related proposals found in the literature that seem to share the most in terms of objectives and followed methods.

6 Alternative MDE approaches for IoT application development

Different MDE approaches targeting the development of IoT solutions have been proposed in the literature. In the following, we report those most related to our proposal by describing and comparing them to the *FloWare* approach, so to finally answer the third research question: "(RQ3)—Which are the benefits of *FloWare* compared to available MDE approaches?".

6.1 Overview of MDE approaches

Above the MDE approaches defined in the literature, ten are strictly linked to supporting the development of IoT applications. We report an overview of their characteristics in Table 1, and then we describe each approach.

Table 1 provides a summary of general aspects for each approach. In particular it reports the source of the approach (column *Source*) and the name of the approach (column *Approach)*. Then, it summarises the kind of tool support that the specific approach make available, as well as the IoT application development phases that the approach support. Considered phases are those of modelling, development, and deployment. Almost all the approaches provide a tool or a library supporting it (column *tool)*. However, not all of them are available for download or testing (column *available*). Nearly all the approaches reference a tool supporting the *modelling phase*, allowing the design of certain kinds of models, such as feature models, using different notations, such as UML, BPMN, and custom DSML (column *models/notations*). Not all the approaches provide support for the *development phase*, meaning that not all of them provide so-called translators or scripts that allow converting models into running code/applications. Those that provide such support provide different target formats such as binary code, Python, Java, JavaScript, C, and GO. Only one of the considered approaches, [25], references the target platforms where to deploy the developed IoT application. All of them except one, [63], validates the approach over an example application scenario. Finally, the *target* of each approach is highlighted, meaning whether it provides support for the development of code that runs directly on devices (*D*) or whether it is focused on the development of software for IoT Applications (*A*).

FRASAD—Framework for sensor application development [63], is a model-driven framework to realise IoT applications targeting different devices in a network. It uses

Table 1 Comparison of the support for the modelling, development and deployment phases of the selected model-driven approaches.

Source	Approach	Tool	Available	Ontology	Modelling phase	Models/notations	Development phase	Target format	Deployment phase	Application scenario	Target
[63]	FRASAD	Yes	–	–	EMF	DSL	Yes	Binary code	–	–	D
[20]	MDE4IoT	Yes	–	–	UML-MARTE	UML	Yes	Java, C++	–	Smart street lights	D
[7]	SPLP-4-IoT	Yes	–	–	CVL tool	CVL Variability Model	–	–	–	Smart shopping centre	D
[15]	Angel Cañete et al.	–	–	–	–	Feature Model	–	–	–	Smart campus	D
[77]	Claudia M. Sosa-Reyna et al.	–	–	–	–	DSML, BPMN, UML	–	–	–	Smart vehicle	D
[58]	ThingML	Yes	Yes	–	EMF	ThingML DSL	Yes	C, Java, JavaScript, GO	–	E-Health	D, A
[62]	AutoIoT	Yes	–	–	–	JSON files	Yes	Python, JavaScript	–	Smart containers	A
[67]	IoTLink	Yes	Yes	–	EMF	DSML	Yes	Java	–	Manufacturing Station	A
[30]	SmartHomeML	Yes	Yes	–	Visual Studio Modelling SDK	DSML	Yes	Platform-specific	Alexa, SmartThings	Smart home	A
[25]	IDeA	Yes	Yes	IoT-A	MBSE Plugin	DSML	–	–	–	Smart building	A
This paper	FloWare	Yes	Yes	IoT-Lite	ADOxx	Feature Model	Yes	JSON	FloWare Platform (Node–RED)	Smart campus	A

D device, *A* application, – not supported

a domain-specific language and rule-based programming to model the IoT application, and a translator to obtain code readable for the different operating systems of a device. It is developed inside the eclipse modelling framework.[14] FRASAD aims to handle and resolve the complexity of IoT application development from the device perspective. In this sense, the authors provide models to describe an IoT application using sensor node domain concepts. Through these modelled concepts, FRASAD aims to improve the reusability, extensibility, and maintainability of a sensor software. This work does not emphasise the actors involved in the multiple steps of its usage (from computation-independent model to platform-independent and finally to platform-specific one), but the authors present results of a user evaluation during which students have been requested to develop an IoT application with FRASAD, and with device operating systems such as TinyOS and ContikiOS. The results show that developing an IoT application with the FRASAD approach is perceived as easier than developing it directly for the device operating systems. However, the approach shows some limitations, especially related to the fact that it only supports TinyOS and ContikiOS operating systems as a target.

MDE4IoT [20] is a model-driven approach to support the architectural modelling and self-adaptation of different device configurations. This approach uses UML models and translators to generate code. The produced code allows the self-adaptation configuration once deployed in the target system. The approach provides a high-level abstraction to support the IoT application development complexity, handling the variability of the inserted elements (devices, physical objects, and others), and supporting reusability of designed artefacts to be deployed on different devices, without considerable modifications to their software functionalities. In doing this, separation of concerns to enable collaborative development is supported. The approach is based on UML-MARTE[15] for modelling and developing real-time embedded systems in Java and C++ languages. The approach is validated using a concrete smart street light case.

SPLP4IOT—software product line process to develop agents for the IoT [7] is a model-driven approach conceived to develop self-managed systems for the IoT domain. The approach aims to enhance the development of self-managed IoT systems based on software agents using software product line models. The reusability is ensured as it is a fundamental principle of the modelling approach used, and the possibility of handling and representing the entire variability of the elements through the common variability language. In this sense, the main scope of this approach is to provide the developer information regarding the configuration of a minimal set of devices supporting the application requirements of the target IoT application, and derive if that configuration can be deployed inside that infrastructure. The approach validation is done by realising a multi-agent system for a smart shopping centre. These models provide the possibility to develop different customised configurations for each shopping experience to satisfy specific requirements.

In [15], the authors propose a model-driven approach that uses a multilayered feature models methodology to capture the variability of IoT software and hardware. The approach provide then the possibility to reuse these models to produce different configurations. The work does not aim to represent the variability concerning an entire IoT system while it aims at ensuring that a derived configuration can be hosted in a specific infrastructure. Specifically, this work represents a support to the developer of edge systems (IoT devices) by providing a modelling approach for cyber-physical systems to reduce the footprint of these technologies. A case study regarding the smart campus domain is provided.

In [77], the authors propose a model-driven architecture to explicitly describe a methodology for designing software applications for the IoT domain. The main focus is on the abstraction levels provided for each phase of developing an IoT Application. The authors propose an architecture composed of four layers based on the software-oriented approach. In this way, the variability in representing device information and the interoperability between the various models are provided. In addition, a set of formal rules to allow the model transformation is provided. A meta-model is given concerning the smart vehicle domain, and the methodology proposed is used to generate code for a real-time system.

ThingML [58] proposes a methodology, a domain-specific modelling language (inside the eclipse modelling framework), and a set of tools to facilitate the collaboration between service developers and platform experts. It is continuously evolving to allow practitioners to have complete control of the code by easily customising compilers on the base of their needs. This personalisation step aims at producing value-added IoT services to be deployed inside devices. In ThingML, the main scope is handling and correctly managing the different communication protocols inside other devices to generate correct artefacts. The artefact produced using this approach can cover various languages, such as Java, JavaScript, C, and GO, allowing to derive code for heterogeneous platforms and devices. An E-Health case study is provided to demonstrate the approach's applicability and to report on scalability and extensibility aspects.

AutoIoT [62] is a model-driven approach based on user-driven choices to generate IoT server-side applications. The approach supports the complex problems of developing IoT applications by providing a high level of abstraction for developing them. AutoIoT allows users to provide a textual representation of the scenario using JSON files. The artefacts produced by the user are translated using specifi-

[14] EMF: https://www.eclipse.org/modeling/emf.

[15] UML-MARTE: https://www.omg.org/omgmarte.

cally designed components (called specialised Builders) to automatically derive the source code of the IoT server-side project. In this sense, this approach does not produce software codes for devices or complete IoT applications. Instead, it only focuses on developing the server-side IoT application part (on Python and JavaScript languages) and does not consider all layers of a complete IoT system development. The approach is validated through a real smart containers scenario involving numerous developers.

IoTLink [67] is a model-driven approach that allows stakeholders with limited programming experience to quickly develop IoT prototype applications. The approach uses a graphical domain-specific language based on a low-code development environment defined within the eclipse modelling framework. This approach first allows the modelling of the elements included inside the application and then to express their behaviour. The modelling elements are provided inside a graphical palette that will enable them to be easily dragged and dropped inside the editor to develop the IoT solution. In this way, the extensibility of the solution and the representation of different elements are favoured. Support for translating the model into Java language is provided. A manufacturing stations case study is provided to ensure the approach's applicability.

SmartHomeML [30] is a model-driven approach that defines a domain-specific modelling language to capture the architecture and specification of a smart home. This work is not portable to other IoT domains, as it was developed specifically for the smart home domain. The main focus of the paper is to provide direct integration between services inside a smart home using Alexa[16] and SmartThings.[17] Model transformation into code artefacts processable by Alexa and SmartThings is obtained through a model-to-text approach. The proposed approach can be extended to provide artefacts for other platforms.

IDeA is a model-driven approach that aims to resolve the complexity of specifying a system model that represents precisely and with a high level of abstraction different hardware and software entities together with their internal information. The IDeA approach provides such abstraction to represent different types of hardware devices, software devices and services through the use of the IoT-A[18] ontology. The authors provide a separation of concerns between all the stakeholders involved in the IoT application development process. In addition, to promote the reusability concept inside the approach, the authors propose using libraries containing reusable model elements for a given domain, such as devices and resources that can be reused by different stakeholders when modelling IoT applications. A smart building scenario demonstrates the applicability of the modelling phase of the IDeA approach.

The *FloWare* approach supports the modelling, development, and deployment phases. In addition, we provide an open-source toolchain to support all the phases. We based *FloWare* on an ontology for the representation of IoT devices and systems. Using a well established ontology allows for describing IoT elements involved in an IoT solution without neglecting important information regarding these elements. For the modelling phase, through the ADOxx metamodelling platform, we developed a feature model library to design these models, and to support the definition of suitable configurations. We also provide translators that permit to derive snippet of flows to be combined in the development phase. These translators allow converting from a standard configuration model file (XML) to a JSON file processable by the low-code development environment. The FloWare tool incorporates the low-code development environment Node-RED to produce directly executable artefacts. We support the IoT developers in storing, managing, and easily extending the automatically generated IoT applications with this tool. *FloWare* supports the development of IoT applications that can retrieve data from the IoT devices involved, elaborate these data, and show them in a customised dashboard based on the inserted devices. We validate the entire approach through a smart campus scenario. Starting from an IoT domain feature model, it is possible to reuse it to produce different configurations (following customer requirements) and, in turn, different IoT applications.

6.2 Comparison of MDE approaches

From the scouting of the approaches in Table 1, it emerged that these approaches have different focuses and targets. In particular, from the target column, it can be seen that some works mainly focus on modelling for a single device (reported as D). In contrast, others provide modelling for an entire IoT application (reported as A). Below we detail the analysis and the comparison between *FloWare* and those emerging approaches that share with our approach the main target, that is providing modelling solutions for entire IoT applications.

Table 2 summarises the results of the conducted analysis and comparison. We categorised and compared the approaches based on the different characteristics that they expose, and in relation to the IoT software development challenges they intend to provide support to. In particular, the source (column *source*) and the name of the approach (column *approach)* are reported. Since the entire IoT application development is an interdisciplinary and complex task, providing a solution to handle the *separation of concerns* principle is considered relevant [11]. We reported this category within the analysis to discuss and compare all the

[16] Alexa: https://www.alexa.amazon.com.

[17] SmartThings: https://www.smartthings.com.

[18] IoT-A: https://www.iot-a.eu.

Table 2 Comparison of model-driven approaches for IoT applications development

Source	Approach	Separation of concerns	Reusability	Variability	Customisation	Extensibility	Scalability
[58]	ThingML	–	–	–	Provided compilers	Generated code	Generated Code
[62]	AutoIoT	–	–	–	–	Generated Code	–
[67]	IoTLink	–	–	–	–	Generated Code	–
[30]	SmartHomeML	–	–	–	–	Additional target platforms	–
[25]	IDeA	Division of stakeholder responsibilities and concerns	IoT model library	–	–	–	–
[this work]	FloWare	Division of stakeholder responsibilities and concerns	Feature Model library	Representation of the entire IoT functionalities and devices	Multiple Feature Model configurations	Provided Library	Provided Library
		Approach division into phases and steps	Generated code		Generated Code	Generated Code	Generated Code

– not supported

approaches based on their possible support to such an aspect. The remaining columns refer to the IoT software development challenges discussed in Sect. 2 (*reusability*, *variability*, and *customisation*). Then, the *extensibility* and *scalability* challenges are reported as emerged from at least one of the analysed works. A critical analysis of all the discussed approaches is provided, detailing for each approach the provided support in addressing the emerged challenges.

Among all the analysed approaches, only in IDeA [25] there is a specific consideration for the *Separation of Concerns* aspect. In particular, the authors provide a specification regarding actors involved in the supported development process, as well as their responsibilities. The intention is to ensure and provide correct mechanisms to address the concerns of the different actors involved. The approach delineates as well a collaborative method to develop IoT applications. Compared to *FloWare*, in addition to specifying each stakeholder's responsibility and concern, we divided the entire approach into distinct phases and steps, where the experts involved are highlighted for each step. All the other approaches do not refer to any development process and do not try to give an intuition of a possible IoT development process in which the approach can better suite. The challenge of developing IoT applications that provide *reusability* potential is highlighted as the main focus by none of the analysed approaches; nonetheless, we considered such an aspect rather relevant for the IoT domain. To maximise its reuse potential, a reusable part must be able to adapt to the needs of a wide variety of users. Those approaches seeking to support this do so by providing a higher level of abstraction to avoid modelling the specific details for a target solution, and providing a tool that allows the modelled application to be easily adapted to other contexts or systems requirements. We noted that only in the IDeA [25] approach reusability can be exploited from a software components perspective. In particular, the approach enables reusability by providing already modelled elements (devices and resources) that could be reused inside different IoT applications. This is a side effect of the developed solution, as their focus does not aim to address such a challenge. Indeed, this functionality permits to share knowledge between different stakeholders when modelling IoT applications. In contrast, we specifically conceived *FloWare* trying to solve the reusability challenge. As a result, reusability is somehow central in all the engineering activities preformed in the context of *FloWare*.

The ability to represent a heterogeneous IoT domain, including all the aspects involved, to be efficiently extended, changed, and customised is defined as *Variability*. In this sense, correctly handling the variability among all the possible changes is fundamental to analyse their correctness before deploying the changes. From the research conducted, no approach addresses the variability challenge as main target of their works. All the reported works provide a modelling

approach that allows to directly model a scenario, not considering a more abstract modelling perspective in which possible points of variability can be inserted to consider a broader scenario. Contrasting, inside *FloWare*, we stress the importance of representing the entire range of variability that such a domain/scenario can represent. Inside our approach, the variability is not focused only on describing an IoT device specific variability aspects. *FloWare* also provides a higher variability degree to manage an entire range of possible IoT applications for a given domain, including all the possible functionalities provided. The representation of these points of variability then makes it possible to provide different decision choices on the base of the considered scenario.

The possibility to handle this variability by representing all the possible functionalities that an IoT system or device can have, so to apply development decisions for different customers, can be defined as *customisation*. In this sense, the customisation is addressed, from a software perspective, in ThingML [58], providing personalised compilers for the customers. *FloWare* addresses the same problem by providing two different customisation perspectives. In our approach, we provide a higher level of abstraction in representing IoT devices and a focus on representing a whole range of IoT solutions for a target IoT domain. In such a way, providing the entire range of possibilities to apply for a given scenario permits customers and experts to customise the target IoT solution based on their requirements and necessities through a model configuration. In addition, we also provide customisation of automatically generated code. Indeed, based on the configurations provided through the functionality models, the artefacts are developed to derive a deployable IoT application. This application has many customisation points that can be chosen in the modelling phase and after obtaining the code.

The issue of providing an approach that can be easily extended (column *extensibility*) is out of the main contributes for *FloWare*. However, as highlighted from the analysed works, it results to be a relevant challenge to face, and on which to compare the different approaches. In some approaches as ThingML [58], AutoIoT [62], and IoTLink [67] the challenge of extensibility is addressed as the possibility of extending the developed code to meet new requirements. In SmartHomeML [30], as conceived as an approach for a specific target solution (smart homes with Alexa and SmartThings platforms incorporated), they provide the possibility to extend the translators provided to include additional target platforms. Following this principle, even if our approach does not aim to solve this challenge, we can certainly assume that *FloWare* can guarantee a certain level of extensibility. Indeed, using *FloWare* it is possible for an enterprise to easily extend and enrich the functionalities provided through the feature models. To the feature models that an organisation has developed, new features can be added as well as the associated devices to provide additional functionalities to different customers. From this point of view, we can infer that the system can be extended easily (mainly using the graphical toolchain we provide). At the same time, as described in Sect. 5, for the IoT application developer it is possible to easily extend the automatic generated IoT application and the provided templates to develop mode complex IoT applications. Clearly, these extensions come with an increased development cost, even though the added characteristics can be successively reused in successive customisation's.

A closely related challenge to the one of extensibility is that of *scalability*. Indeed, scalability can be related to measuring performance and costs of an approach in relation to is capability to respond to increasing or decreasing specific needs [38]. In ThingML [58] the scalability is discussed concerning the increase or decrease in device performance, using benchmarks to provide values as devices energy consumption and execution time. As with extensibility, our *FloWare* does not directly address this challenge as the approach mentioned above does. However, it is reasonable to assume that, from an IoT domain modelling perspective, our approach can ensure a certain degree of scalability, as it allows representing a vast scenario, including many different functionalities and related devices, as demonstrated in the smart campus scenario in Sect. 5. We demonstrate that many other configurations can be derived and translated into deployable code artefacts without increasing time and cost, even from a large application scenario domain.

Compared to the identified approaches, our *FloWare* stands out for the capability to deal with almost all the identified challenges for the development of an IoT Application and for the completeness of the toolchain capable of supporting all the phases of the MDE approach, from modelling of an IoT domain to the actual deployment of an IoT application. In addition, *FloWare* differs from being strongly based on the separation of concerns principle, an almost absent principle for what concerns the other approaches, clarifying and pointing out possible different phases and steps necessary to develop an IoT application, and then identifying specific expertise involved in applying the entire approach.

7 Limitations and extensibility

In this research work, we presented the *FloWare* approach and its supporting toolchain. They provide support to all the stages involved in the development of an IoT solution. Nevertheless, our approach and toolchain present some limitations.

Referring to the *FloWare* approach, some limitations are linked to the management of large IoT domains, which may incorporate several IoT systems and sub-system. A possible solution to this issue could be that of defining modular feature

models as proposed in [8]. In particular, the models should be divided into parts which are hierarchically arranged to form a complete feature model. The modular feature model representation can enhance an additional reusability and feature model evolution. In addition, in *FloWare* models must be maintained to reflect the enterprise's capability properly. Therefore, models can and must be updated to support the evolution of a domain. This also affects the proposed toolchain that needs to be updated to support added IoT Systems and IoT devices. In particular, it is possible to integrate new information regarding IoT devices by modifying the library provided within the metamodelling platform. For example, extending the list of protocols used by adding a specific one under the Communication Protocol category is possible. This type of change also impacts the translators for the target platforms, which must be updated to process the new information correctly.

For what concerns the *FloWare* toolchain, it provides IoT developers with a selection of templates for the application logic, so as to simplify and speed up the development of IoT applications. However, up to now, the selection of templates is limited and represents only basic examples of application logic. An enterprise that uses our approach could reasonably define more complex templates based on its experience, and already implemented best practices. In addition, our toolchain can only support translating feature model configurations from the XML format into the JSON format that can be processed by the Node-RED tool (which is integrated into our FloWare Platform). To overcome this limitation, we could develop ad hoc translators for additional target platforms, and add them to the ADOxx library, or establish a standard format based on the IoT-Lite ontology structure used, and leave the translation into specific formats to interested developers.

8 Conclusions and future work

To solve the inherent complexity of modelling and developing IoT applications, we presented the *FloWare* approach. The idea is to combine and exploit the potential of a model-driven engineering approach through feature models and Low Code Development. These models allow handling the huge *variability* dimension typical of the IoT domain, and enable the *reuse* of *knowledge* in the production of different deployment solutions for IoT applications. In this sense, the *FloWare* supports a high degree of *customisation* of generic IoT solutions to derive solutions that satisfy the needs of differentcustomers in different ways. Derived variations can be translated into code and deployed inside a low-code development environment. The approach provides a way to lighten the overall complexity of IoT application development by applying the separation of concern principle and providing support for the various development phases, from the IoT modelling to the application development.

The paper reported on how separating the modelling and configuration of different IoT systems, from the activities of IoT application developers, allows developers to stay focused on defining a proper application logic instead of having directly to deal with IoT device's technicalities. With our solution, the developers can remain unaware of each device technical specifications, while he/she will inherit such information from the feature model configurations that IoT and modelling experts have defined. In addition, feature models emphasise *reusability* as their intrinsic property, and they allow modelling the entire IoT domain solution once and then reusing that model to provide different configurations, and then solutions. The models are proposed following the IoT-Lite ontology to represent all the elements concerning the IoT context. They can be easily extended to satisfy scalability without impacting the required effort. In addition, these models permit the better management of *variability* aspects, that are a distinctive characteristic of the IoT domain. This variability is increased by the *customised* requests the customers can make for the various solutions.

The paper illustrated the application of the *FloWare* approach and the relative toolchain, applying it to a realistic scenario in the smart campus domain. The supporting toolchain leverages the usage of the ADOxx metamodelling platform through a Feature Model library developed for the modelling of feature models, their configuration selection, up to their translation into code artefacts in the form of basic templates executable by a proper target platform. To support the development of IoT applications, we also provided the FloWare Platform that permit to automatically imports these artefacts, properly configures the development environment, and allows the IoT developer to extend the basic templates already provided to realise complex IoT applications.

In future work, we plan to involve other IoT and modelling experts to design and develop different IoT systems in practice, and to validate and test the approach in different scenarios. Referring to the set of templates representing basic application logic, we plan to extend it to include more complex application logic that can simplify and speed up the development of IoT applications.

Acknowledgements This work has been partially supported by Marche Region in implementation of the financial programme POR MARCHE FESR 2014–2020, project "Miracle" (Marche Innovation and Research fAcilities for Connected and sustainable Living Environments), CUP B28I19000330007, and by the MIUR project PRIN "Fluidware" (A Novel Approach for Large-Scale IoT Systems, n. 2017KRC7KT).

References

1. Al-Masri, E., Kalyanam, K.R., Batts, J., Kim, J., Singh, S., Vo, T., Yan, C.: Investigating messaging protocols for the internet of things (IoT). IEEE Access **8**, 94880–94911 (2020)
2. Alghamdi, A., Shetty, S.: Survey Toward a Smart Campus Using the Internet of Things. In: 4th IEEE International Conference on Future Internet of Things and Cloud, pp. 235–239. IEEE Computer Society, FiCloud (2016)
3. Amadeo, M., Campolo, C., Quevedo, J., Corujo, D., Molinaro, A., Iera, A., Aguiar, R.L., Vasilakos, A.V.: Information-centric networking for the internet of things: challenges and opportunities. IEEE Netw. **30**(2), 92–100 (2016)
4. Amaral, L., de Matos, E., Tiburski, R., Hessel, F., Tessaro Lunardi, W., Marczak, S.: Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G, pp. 333–367. Springer, Cham (2016)
5. Appel, S., Kleber, P., Frischbier, S., Freudenreich, T., Buchmann, A.: Modeling and execution of event stream processing in business processes. Inf. Syst. **46**, 140–156 (2014)
6. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
7. Ayala, I., Amor, M., Fuentes, L., Troya, J.M.: A software product line process to develop agents for the IoT. Sensors **15**, 15640–15660 (2015)
8. Bagheri, E., Ensan, F., Gasevic, D., Boskovic, M.: Modular feature models: representation and configuration. J. Res. Pract. Inf. Technol. **43**(2), 109–140 (2011)
9. Beek, M.H.T., Cledou, G., Hennicker, R., Proença, J.: Featured team automata. In: Formal Methods—24th International Symposium, Lecture Notes in Computer Science, vol. 13047, pp 483–502. Springer, Berlin (2021)
10. Bermúdez-Edo, M., Elsaleh, T., Barnaghi, P.M., Taylor, K.L.: IoT-Lite: a lightweight semantic model for the Internet of Things. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, 2016, pp 90–97. IEEE Computer Society (2016)
11. Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Conallen, J., Houston, K.A.: Object-oriented analysis and design with applications, third edition. ACM SIGSOFT Softw. Eng. Notes **33**(5) (2008)
12. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**(1), 5–13 (2020)
13. Buchmann, T., Westfechtel, B.: Mapping feature models onto domain models: ensuring consistency of configured domain models. Softw. Syst. Model. **13**(4), 1495–1527 (2014)
14. Caracas, A., Kramp, T.: On the expressiveness of BPMN for modeling wireless sensor networks applications. In: Business Process Model and Notation—Third International Workshop, BPMN, Lecture Notes in Business Information Processing, vol. 95, pp. 16–30. Springer, Berlin (2011)
15. Cañete, A., Amor, M., Fuentes, L.: Supporting IoT applications deployment on edge-based infrastructures using multi-layer feature models. J. Syst. Softw. **183**, 111086 (2022)
16. Chagnon-Lessard, N., Gosselin, L., Barnabé, S., Bello-Ochende, T., Fendt, S., Goers, S., da Silva, L.C.P., Schweiger, B., Simmons, R., Vandersickel, A., Zhang, P.: Smart campuses: extensive review of the last decade of research and current challenges. IEEE Access **9**, 124200–124234 (2021)
17. Chebudie, A.B., Minerva, R., Rotondi, D.: Towards a definition of the Internet of Things (IoT). IEEE Internet Initiat. **1**, 1–86 (2015)
18. Chen, S., Xu, H., Liu, D., Hu, B., Wang, H.: A vision of IoT: applications, challenges, and opportunities with china perspective. IEEE Internet Things J. **1**(4), 349–359 (2014)
19. Chiu, H.H., Wang, M.S.: A study of IoT-aware business process modeling. Int. J. Model. Optim. **3**(3), 238 (2013)
20. Ciccozzi, F., Spalazzese, R.: MDE4IoT: Supporting the Internet of Things with model-driven engineering. In: Intelligent Distributed Computing X—Proceedings of the 10th International Symposium on Intelligent Distributed Computing—IDC 2016, Studies in Computational Intelligence, vol. 678, pp. 67–76 (2016)
21. Ciccozzi, F., Crnkovic, I., Di Ruscio, D., Malavolta, I., Pelliccione, P., Spalazzese, R.: Model-driven engineering for mission-critical IoT systems. IEEE Softw. **34**(1), 46–53 (2017)
22. Compagnucci, I., Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: Modelling notations for IoT-aware business processes: a systematic literature review. In: Business Process Management Workshops—BPM 2020 International Workshops, Lecture Notes in Business Information Processing, vol. 397, pp. 108–121. Springer, Berlin (2020)
23. Corradini, F., Fedeli, A., Fornari, F., Polini, A., Re, B.: FloWare: an approach for IoT support and application development. In: Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, vol. 421, pp. 350–365. Springer, Berlin (2021)
24. Corradini, F., Fedeli, A., Fornari, F., A.P.: X-IoT: a model-driven approach for cross-platform IoT applications development. In: SAC'22: The 37th ACM/SIGAPP Symposium on Applied Computing, pp. 1448–1451. ACM (2022)
25. Costa, B., Pires, P.F., Delicato, F.C.: Modeling IoT applications with SysML4IoT. In: 42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, pp. 157–164. IEEE Computer Society (2016)
26. da Cunha, C., Agard, B., Kusiak, A.: Design for cost: module-based mass customization. IEEE Trans. Autom. Sci. Eng. **4**(3), 350–359 (2007)
27. De Cremer, D., Nguyen, B., Simkin, L.: The integrity challenge of the internet-of-things (IoT): on understanding its dark side. J. Mark. Manag. **33**(1–2), 145–158 (2017)
28. Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: two sides of the same coin? Softw. Syst. Model. **21**(2), 437–446 (2022)
29. Dong, Z.Y., Zhang, Y., Yip, C., Swift, S., Beswick, K.: Smart campus: definition, framework, technologies, and services. IET Smart Cities **2**(1), 43–54 (2020)
30. Einarsson, A.F., Patreksson, P., Hamdaqa, M., Hamou-Lhadj, A.: Smarthomeml: towards a domain-specific modeling language for creating smart home applications. In: IEEE International Congress on Internet of Things, ICIOT, pp. 82–88. IEEE Computer Society (2017)
31. Farhan, L., Kharel, R., Kaiwartya, O., Quiroz-Castellanos, M., Alissa, A., Abdulsalam, M.: A concise review on Internet of Things (IoT)-problems, challenges and opportunities. In: 2018 11th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP), pp. 1–6. IEEE (2018)
32. Feljan, J., Karapantelakis, A., Mokrushin, L., Inam, R., Fersman, E., Azevedo, C., Raizer, K., Souza, R.: Tackling IoT complexity. Ericsson Rev. (Engl. ed) **95**(2), 60–69 (2017)

33. Fortino, G., Guerrieri, A., Russo, W., Savaglio, C.: Middlewares for smart objects and smart environments: Overview and comparison. In: Internet of Things Based on Smart Objects, Technology, Middleware and Applications, pp. 1–27. Springer, Berlin (2014)
34. 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)
35. Gill, A.Q., Behbood, V., Ramadan-Jradi, R., Beydoun, G.: IoT architectural concerns: a systematic review. In: Proceedings of the Second International Conference on Internet of things and Cloud Computing, ICC'17. Association for Computing Machinery (2017)
36. Grefen, P., Brouns, N., Ludwig, H., Serral, E.: Co-location specification for IoT-aware collaborative business processes. In: Information Systems Engineering in Responsible Information Systems, vol. 350, pp. 120–132. Springer, Berlin (2019)
37. Gregor, S., Hevner, A.R.: Positioning and presenting design science research for maximum impact. MIS Q **37**(2), 337–355 (2013)
38. Gupta, A., Christie, R., Manjula, P.: Scalability in internet of things: features, techniques and research challenges. Int. J. Comput. Intell. Res. **13**(7), 1617–1627 (2017)
39. Gupta, B.B., Quamara, M.: An overview of internet of things (IoT): architectural aspects, challenges, and protocols. Concurr. Comput. Pract. Exp. **32**(21), e4946 (2020)
40. Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., Reinfurt, L.: A detailed analysis of IoT platform architectures: concepts, similarities, and differences. In: Internet of Everything, pp. 81–101. Springer, Berlin (2018)
41. Hasić, F., Asensio, E.S.: Executing IoT processes in BPMN 2.0: current support and remaining challenges. In: 2019 13th International Conference on Research Challenges in Information Science (RCIS), pp. 1–6 (2019)
42. 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)
43. Hussain, M., et al.: Internet of things: challenges and research opportunities. CSI Trans. ICT **5**(1), 87–95 (2017)
44. Ihirwe, F., Indamutsa, A., Di Ruscio, D., Mazzini, S., Pierantonio, A.: Low-code engineering for Internet of Things: a state of research. In: MODELS'20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, 2020, pp. 74:1–74:8. ACM (2020)
45. Ihirwe, F., Indamutsa, A., Di Ruscio, D., Mazzini, S., Pierantonio, A.: Cloud-based modeling in IoT domain: a survey, open challenges and opportunities. In: Conference: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS 2021), pp. 73–82 (2021a)
46. Ihirwe, F., Indamutsa, A., Ruscio, D.D., Mazzini, S., Pierantonio, A.: Cloud-based modeling in IoT domain: a survey, open challenges and opportunities. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, pp. 73–82. IEEE (2021b)
47. 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)
48. 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)
49. ...Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burattin, A., Di Ciccio, C., Fortino, G., Gal, A., Kannengiesser, U., Leotta, F., Mannhardt, F., Marrella, A., Mendling, J., Oberweis, A., Reichert, M., Rinderle-Ma, S., Serral, E., Song, W., Su, J., Torres, V., Weidlich, M., Weske, M., Zhang, L.: The internet of things meets business process management: a manifesto. IEEE Syst. Man Cybern. Mag. **6**(4), 34–44 (2020)
50. Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE Softw. **19**(4), 58–65 (2002)
51. Kazmi, A., Jan, Z., Zappa, A., Serrano, M.: Overcoming the heterogeneity in the internet of things for smart cities. In: Podnar Žarko, I., Broering, A., Soursos, S., Serrano, M. (eds.) Interoperability and Open-Source Solutions for the Internet of Things, pp. 20–35. Springer, Cham (2017)
52. Lee, I.: The Internet of Things for enterprises: an ecosystem, architecture, and IoT service business model. Internet of Things **7**, 100078 (2019)
53. Lee, I., Lee, K.: The Internet of Things (IoT): applications, investments, and challenges for enterprises. Bus. Horiz. **58**, 431–440 (2015)
54. Li, W.: Design of smart campus management system based on Internet of Things technology. J. Intell. Fuzzy Syst. **40**(2), 3159–3168 (2021)
55. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development: the practitioners' perspective. In: Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11 (2021)
56. Meyer, S.: Internet of Things architecture IoT—a project deliverable D2.2—concepts for modelling IoT-aware processes. In: Technical report, VDI/VDE Innovation and Technik GMBH, Germany, 2012. EC FP7 IoT-A (2012)
57. Meyer, S., Ruppen, A., Hilty, L.M.: The things of the internet of things in BPMN. In: Advanced Information Systems Engineering Workshops—CAiSE, Lecture Notes in Business Information Processing, vol. 215, pp. 285–297. Springer, Berlin (2015)
58. Morin, B., Harrand, N., Fleurey, F.: Model-based software engineering to tame the IoT jungle. IEEE Softw. **34**(1), 30–36 (2017)
59. Morrison, J.P.: Flow-Based Programming, 2nd Edition: A New Approach to Application Development., CreateSpace (2010)
60. Mosterman, P.J., Zander, J.: Industry 4.0 as a cyber-physical system study. Softw. Syst. Model. **15**(1), 17–29 (2016)
61. Muhamad, W., Kurniawan, N.B., Suhardi, Yazid, S.: Smart campus features, technologies, and applications: a systematic literature review. In: 2017 International Conference on Information Technology Systems and Innovation (ICITSI), pp. 384–391 (2017)
62. 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. ACM (2020)
63. Nguyen, X.T., Tran, H.T., Baraki, H., Geihs, K.: FRASAD: a framework for model-driven IoT application development. In: 2nd IEEE World Forum on Internet of Things, pp. 387–392. IEEE Computer Society (2015)
64. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. J. Syst. Softw. **103**, 62–84 (2015)
65. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manag. Inf. Syst. **24**(3), 45–77 (2008)
66. 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)
67. Pramudianto, F., Kamienski, C.A., Souto, E., Borelli, F.F., Gomes, L.L., Sadok, D., Jarke, M.: IoT Link: an Internet of Things prototyping toolkit. In: 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops, 2014, pp. 1–9. IEEE Computer Society (2014)

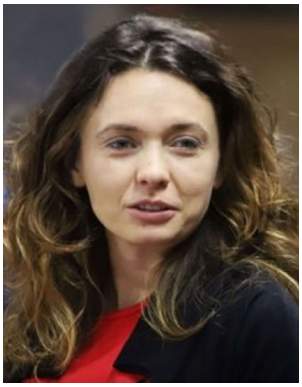
68. Raja, S.P., Rajkumar, T.D., Raj, V.P.: Internet of things: challenges, issues and applications. J. Circuits Syst. Comput. **27**(12), 1830007:1–1830007:16 (2018)
69. Ranganathan, A., Al-Muhtadi, J., Chetan, S., Campbell, R., Mickunas, M.D.: Middlewhere: a middleware for location awareness in ubiquitous computing applications. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 397–416. Springer, Berlin (2004)
70. Reinhartz-Berger, I., Figl, K., Haugen, Ø.: Investigating styles in variability modeling: hierarchical vs. constrained styles. Inf. Softw. Technol. **87**, 81–102 (2017)
71. Sahay, A., Indamutsa, A., Ruscio, D.D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, 2020, pp. 171–178. IEEE (2020)
72. Schmidt, D.C.: Model-driven engineering. Comput.-IEEE Comput. Soc. **39**(2), 25 (2006)
73. Serrano, M., Gyrard, A., Tragos, E.Z., Dang, H.N.: FIESTAIoT project: federated Interoperable Semantic IoT/cloud Testbeds and Applications. In: Companion of the The Web Conference, pp. 425–426. ACM (2018)
74. 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)
75. Sicari, S., Rizzardi, A., Coen-Porisini, A.: Smart transport and logistics: a node-red implementation. Internet Technol. Lett. **2**(2) (2019b)
76. Sivrikaya, F., Sassi, N.B., Dang, X., Görür, O., Kuster, C.: Internet of smart city objects: a distributed framework for service discovery and composition. IEEE Access **7**, 14434–14454 (2019)
77. 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)
78. Steinmetz, C., Schroeder, G., dos Santos, Roque A., Pereira, C.E., Wagner, C., Saalmann, P., Hellingrath, B.: Ontology-driven IoT code generation for FIWARE. In: 15th IEEE International Conference on Industrial Informatics, INDIN, pp. 38–43. IEEE (2017)
79. Udoh, I.S., Kotonya, G.: Developing IoT applications: challenges and frameworks. IET Cyper. Phys. Syst. Theory Appl. **3**(2), 65–72 (2018)
80. Valderas, P., Torres, V., Serral, E.: Modelling and executing IoT-enhanced business processes through BPMN and microservices. J. Syst. Softw. **184**, 111139 (2022)
81. Vanhoorelbeke, F., Snoeck, M., Serral, E.: Identifying the challenges and requirements of enterprise architecture frameworks for IoT systems. In: Research Challenges in Information Science, pp. 576–581. Springer, Berlin (2020)
82. Vuppalapati, C.: Building Enterprise IoT Applications, 1st edn. CRC Press, Boca Raton (2019)
83. Wehlitz, R., Jauer, F., Rößner, I., Franczyk, B.: Increasing the reusability of iot-aware business processes. In: FedCSIS (Position Papers), pp. 17–22 (2020)
84. Yousfi, A., Batoulis, K., Weske, M.: Achieving business process improvement via ubiquitous decision-aware business processes. ACM Trans. Internet Technol. **19**(1), 14:1–14:19 (2019)
85. Zdravkovic, J., Svee, E., Giannoulis, C.: Capturing consumer preferences as requirements for software product lines. Requir. Eng. **20**(1), 71–90 (2015)
86. Zdravković, M., Zdravković, J., Aubry, A., Moalla, N., Guedria, W., Sarraipa, J.: Domain framework for implementation of open IoT ecosystems. Int. J. Prod. Res. **56**(7), 2552–2569 (2018)
87. Zhai, X., Dong, Y., Yuan, J.: Investigating learners' technology engagement—a perspective from ubiquitous game-based learning in smart campus. IEEE Access **6**, 10279–10287 (2018)

Flavio Corradini is Full Professor of Computer Science at the University of Camerino. He received a Laurea Degree in Computer Science from the University of Pisa and a PhD in Computer Engineering from the University of Rome "La Sapienza". He has been Director of the Mathematics and Computer Science Department (2006–2009), President of the center for digital services and information systems of the University of Camerino (2004–2010), Coordinator of the Computer Science Studies of the University of Camerino (2004–2006), and rector of the University of Camerino (2010–2016). His main research activities are in the area of formal specification, verification of concurrent, distributed and real-time systems.

Arianna Fedeli is a Ph.D student at the University of Camerino (UNICAM), since 2020. She graduated cum laude with a master's degree in Computer Science and cum laude with a bachelor of science degree in Informatics, both from the University of Camerino. Currently, she is a member of the PROcesses and Services Laboratory (PROS Lab), where she conducts her activities. Her research interests refer to the area of Model-Driven Engineering applied to the Internet of Things (IoT) domain. Particular attention is paid to applying software engineering methodologies to manage variability and customisation in developing IoT applications and the investigation of applying these methodologies to emerging technology solutions such as IoT cloud platforms, Fog computing and Low-Code Development. In addition, her research interests also range in the digital twins field, scouting modelling solutions for such a domain.

Fabrizio Fornari is a Research Fellow in Computer Science at the University of Camerino, Italy, since 2022. He obtained his Ph.D. (2018) and M.Sc. (2013) in Computer Science at the University of Camerino. From 2017, he is a member of the PROcesses and Services Laboratory (PROS Lab), within which he conducts his research activities. His research activities focus on the management of business processes starting from their modeling to the analysis of properties passing through the definition of solutions based on formal methods, therefore with a solid foundation of theoretical computer science. In recent years, he started investigating how the Internet of Things can affect business processes and how model-driven engineering can be applied to support the development of IoT and digital twin solutions.

Andrea Polini is an Associate Professors at University of Camerino (UNICAM). His research interests are in the area of Software Engineering in general, and in particular on Modeling methods and Quality Assurance strategies for Complex Software Systems. He is a coordinating member of the PROS Lab at UNICAM. Before joining UNICAM he was a researcher at ISTI-CNR in Pisa, and he got a PhD in Computer Engineering from Scuola Superiore Sant'Anna in Pisa. His research has been always linked to his participation in many EU research projects, and he has been Project Scientific Leader for the EU Collaborative Project Learn PAd. He is the co-author of more than 100 scientific publications.



Barbara Re is Associate Professor of Computer Science at the University of Camerino (UNICAM). She received her PhD in Information Science and Complex System from University of Camerino. Her research interests refer to the area of Business Process Management from modelling to analysis. Particular attention is paid to push the use of formal methods as methodological and automatic tools for the development of high-quality process-aware information systems. She was involved in multidisciplinary research projects in collaboration with national and international research institutes and companies.