Model Based Methodology and Framework for Design and Management of Next-Gen IoT Systems

Xu Tao, Davide Conzon, Enrico Ferrera
LINKS Foundation
Turin, Italy
{name.surname}@linksfoundation.com

Shuai Li
CEA, LIST
Paris-Saclay, France
{name.surname}@cea.fr

Juergen Goetz
Siemens AG Corporate Technology
Munich, Bavaria, Germany
juergen.goetz@siemens.com

Laurent Maillet-Contoz,
Emmanuel Michel, Mario Diaz-Nava
STMicroelectronics
Grenoble, France
{firstname.lastname}@st.com

AbdelHakim Baouya, Salim Chehida
Univ. Grenoble Alpes
Grenoble, France
{name.surname}@univ-grenoble-alpes.fr

CEUR-WS.org/Vol-2739/paper_11.pdf

***Abstract*—Internet of Things (IoT) is a pervasive technology covering many applications areas (Smart Mobility, Smart Industry, Smart Healthcare, Smart Building, etc.). Its success and the technology evolution allow targeting more complex and critical applications such as the management of critical infrastructures and cooperative service robotics, which requires real time operation and a higher level of intelligence in the monitoring-control command for decision-making. Furthermore, these applications type need to be fully validated in advance considering that bugs discovered during real operation could cause significant damages. In order to avoid these drawbacks, IoT developers and system integrators need advanced tools and methodologies. This paper presents a methodology and a set of tools, defined and developed in the context of the BRAIN-IoT European Union (EU) project. The overall framework includes both Open semantic models to enforce interoperable operations and exchange of data and control features; and Model-based development tools to implement Digital Twin solutions to facilitate the prototyping and integration of interoperable and reliable IoT system solutions. After describing the solution developed, this paper also presents concrete use cases based on the two critical systems mentioned above, leveraging the application scenarios used to validate the concepts developed and results obtained by the BRAIN-IoT project.**

***Index Terms*—Model-Based System Engineering, Internet of Things, Digital Twin, Brain-IoT**

I. Introduction

Nowadays, the IoT concept is adopted in new application domains, allowing fast digitalization of contemporary society [1]. The application of IoT in innovative scenarios such as critical infrastructures management, and cooperative service robotics demand to satisfy a set of strict requirements in term of low latency, high reliability, adaptability, heterogeneity and scalability, highly more challenging than the ones required by the traditional (e.g., domotics) environments. To satisfy these requirements, the IoT developers needs to introduce in their solutions next generation Internet of Things (next-gen IoT) technologies, e.g., Edge Computing, Artificial Intelligence (AI), Digital Twin, among others [2], thus leading them to become more complex to design and manage and requiring the introduction of methodologies and tools that ease the users their development and runtime management.

Recently, several approaches have been proposed to ease the development of IoT systems (see section III-A). However, these solutions do not generally support all the functionalities required by next-gen IoT applications and focus only on development, not supporting the other phases of the application life-cycle, e.g., deployment, validation, monitoring and adaptation at runtime. Currently, the market asks for IoT solutions supporting business critical tasks that can be deployed rapidly and with low costs. Such solutions need to allow the design of applications involving several interconnected heterogeneous platforms and smart things and, at the same time, be able to deploy, monitor and evolve the designed complex solution adapting automatically and at runtime to environmental changes.

This paper is organized as follow: Section II presents the motivation and the requirements that has led to the development of the BRAIN-IoT Modeling & Verification Framework presented in this work. In Section III-A, some existing model-based system engineering approaches are discussed. Then, the BRAIN-IoT modeling methodology is introduced in Section IV and the implementation of the methodology is illustrated in Section V. Next, two use cases, in Section VI, are exploited to demonstrate the functionalities provided by the BRAIN-IoT Modeling & Verification Framework. Finally, Section VII concludes the paper.

II. Motivation and Requirements

The design of an IoT system today presents considerable complexity. Several factors contribute to this complexity: first, an understanding of IoT is still too focused on IoT technologies and objects (device types, communication protocols, cloud, database type, etc.). Such a vision loses sight of the true purpose of the system to develop and leads most of the time to unsatisfactory solutions. Then, the wide variety of IoT systems, in terms of their deployment, is a technological

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

source of complexity: IoT systems can be considered with a ”local” deployment such as an automatic lighting system of a house (measurement, storage, data analysis and execution of the application on a single ”device” connected to the owner’s smartphone), or systems with a ”highly distributed” architecture such as a weather forecasting system (sensor networks for data capture capabilities, plus a cloud for storing, analyzing data and running the end-user application). Another aspect concerns the need to integrate legacy systems, i.e., IoT systems not designed from “zero”, the problem is then to create new innovative services on the basis of existing infrastructures. In this case, it is necessary to ”connect” what exists and then develop / integrate supporting components (authentication, monitoring, data distribution, data analysis, etc.). The objective of this work is to develop a framework that supports the designer of complex next-gen IoT applications, easing the use of disruptive technologies, e.g., Digital Twins.

For this scope, the solution proposed needs to allow i) modelling several aspects of an IoT system: the physical layer, the IoT devices, the system layer, the system behaviors and the interactions among the components. ii) Composing IoT services also provided by different IoT platforms, using a model-based design approach and semantically annotated data formats to support interoperability among heterogeneous systems. iii) Formally verifying and validating the models designed with the framework. iv) The automatic generation of code from the models to be deployed on real devices. v) Supporting the co-simulation approach, with the creation of a mixed reality environment, where virtual and real entities can interact with each other. vi) Monitoring the IoT applications at run-time, keeping the models and the physical world synchronized with each other. The next section will provide a state-of-the-art (sota) of the solutions available to design and manage next-gen IoT applications.

III. Background

A. SOTA

A system design process that adheres to the principles and methods related to ”system engineering” allows to understand the design phase of a complex system w.r.t. expected functionality, cost, time, and quality. In the area of critical systems, system engineering is in full development and benefits from domain-specific and often user-specific solutions. Model-based system engineering is a strong trend, and recent projects such as AGeSys [3] and Connexion [4] highlight its importance and have provided an important foundation for the development of tooled solutions. This work places particular emphasis on the importance of having interoperable, open, and scalable solutions.

In the field of IoT, however, the offer has not yet reached this level of maturity, even though stakeholders are investing heavily in the implementation of model-based design solutions for distributed software architectures, such as Ericsson for network software through its development solution based on the Papyrus [5] open source tool; or THALES through the establishment of its Capella [6] engineering chain; or Dassault System with its Delmia [7] solution.

More specifically, in the field of IoT system engineering, the Internet of Things - Architecture (IoT-A) project [8] proposes a methodological framework of design based on the elaboration of a set of models for the specification of the system according to different views. It is important to note that the reference models proposed by IoT-A are independent of modeling languages that could be used to represent them, even though different case studies applying this methodology have been developed using Unified Modelling Language (UML) models.

In the world of standards, the international standardization organization Object Management Group (OMG) has developed the System Modeling Language (SysML) [9] with the aim of treating software systems, CPS or organizational systems alike. The generality of SysML requires that it be specialized and, like any language, it must also be accompanied by a specific methodology for each application domain so that its use is best adapted to a given field of application and reach a level of maximum efficiency. This is for example what has been achieved for the field of critical embedded systems in the AGeSys project, or for the field of nuclear power plants control-command systems in the project Connexion.

Like embedded real-time systems, execution platforms are a prime concern for IoT systems. The problem of platform modeling and application allocation on the execution platforms, in the field of embedded real-time systems, has been addressed by the OMG in the context of the Modeling and Analysis of Real-Time and Embedded systems (MARTE) [9] standard, which complements SysML on this aspect. However, MARTE needs to be taken up for two aspects: the first is related to its richness which makes it a complex language to handle and requires to be supported by a methodological approach targeting the essential elements to the modeling of IoT systems; the second point is that it remains very focused on the embedded concerns, and only addresses very little distributed system aspects, in particular the aspects related to the communications and the interactions between the distributed components. The link between the system / platform model, and the component / communication standards, needs to be refined and even developed for some new cases, e.g., OMG CORBA Component Model (CCM) [10], DDS for Lightweight CCM (DDS4CCM) [11], Service oriented Architecture Modeling Language (SoAML) [12] and more recently Unified Component Model (UCM) [13]).

The authors have identified that no one of the approaches described above is able to satisfy all the requirements listed in Section II. Besides, methodologies proposed for the IoT domain today do not benefit from dedicated model-based development tools. Indeed, either these methodologies target closed systems (e.g. critical embedded systems) or their only goal is to only establish a guideline for the design process. In the latter case, neither languages, nor development techniques, are defined. However, the choice of a modelling language, and its dedicated tools that will exploit the models, is imperative for the development of well-suited integrated development environment for the specific needs of the IoT domain.

B. Progrees Beyond SOTA

In this paper, the authors propose system engineering solution for the IoT domain. The proposed solution combines standard languages with methodologies, while remaining open to domain-specific practices. Furthermore, the approach fosters tools related to the modelling languages and methodologies. The authors of this work believe this solution aims at filling the gaps in the current system engineering offering for IoT systems, while not omitting current developer practices.

The solution provides such main features: i) Modelling methodology: A lightweight methodology accompanies IoT Modelling Language (IoT-ML). Using the language's capacity to describe different levels of abstraction, the methodology proposes to model entities representing system, software, and hardware components. By relating such components among them, and by linking them to their domain-specific representation and tools, the authors of this paper promote a system holistic view of the whole IoT solution. ii) System behavior modeling: It is based on IoT-ML and integrated with World Wide Web Consortium (W3C) Web of Things (WoT) Thing Description (TD), thus the service provided by the IoT devices, communication protocol and its data structure can be modeled as well. iii) IoT physical layer modeling and validation: It models the IoT devices, BRAIN-IoT adopted the digital twin concept to provide the ability for system application validation on the modeled IoT devices before deploying in the production environment. iv) Models@Runtime: One particular aspect that is not well explored in the IoT domain, and critical embedded systems domain altogether, is the human-friendly monitoring of system state at runtime, directly through the system models. A model-based runtime monitoring approach usually helps identify and fix deviations observed at runtime, compared to formal specifications defined in the models. In the BRAIN-IoT solution, not only is runtime formal validation a priority, but the authors also wish to benefit from monitoring to enable behavior explanations friendly to humans. v) Code generation: The code to be deployed is automatically generated from the models.

As mentioned, the core of the system engineering solution proposed for IoT is based on IoT-ML and its Papyrus tooling. However, as a reminder, this work wishes to promote integration of domain practices through linking artefacts or refining models. The following sections describe some domain-specific languages and tools that refine entities in the system model.

IV. BRAIN-IoT MODELING METHODOLOGY

This section will present the modelling methodology proposed BRAIN-IoT. As the primary objective, BRAIN-IoT aims to allow modeling in different abstraction layers. Firstly, it designs the system-level model composed by the involved components' functionalities and interactions based on the system requirements. The system model also eases the linking towards real devices and external services through metadata generation, and human-friendly monitoring of device behaviors. Then, the system-level model will be refined as a formal software-level model whose correctness will be checked by using the statistic modeling checking and formal verification. The obtained correct software model is used by a code generator to generate the software artefacts. Finally, IoT devices can be modeled as the refined physical-level models to validate and test the IoT applications before deploying to the real physical infrastructure. Hence, it allows to have three different abstraction level models, including the system-level architecture models, software-level components models, and physical-level IoT devices, the focus of each abstraction level is as follows: i) **system-level models** focuses on composability of services provided by the IoT devices/platforms and the overall system behaviors. The system model also serves as an aggregator of blocks that are refined in lower level models, i.e., the software and device described further. Finally, the authors of this paper promote exploitation of the system model, not just for design, but also to help deployment, fast prototyping, and human-friendly monitoring of behaviors at runtime. ii) **software-level models** are the formal models of computation with their formal validation capabilities, and their runtime to guarantee execution conformity to formal specifications, are obtained from the system-level models through the syntactical transformation. iii) **IoT physical device models** allow the virtual representation of the edge domain of an IoT system following a Digital Twin approach, i.e., the possibility to combine a virtual world with a physical one to validate the behavior and performances of a complete system or a system of systems in various steps. This is necessary when the system is complex (the number of devices is high, from 100's to 1000's) and/or in the case of critical systems. These models allow the modeling and simulation of the components performing the sensing / actuating, computing and communication functions, which are the fundamental functions of an IoT system. This approach could also facilitate the carry-out of data analysis. However, to design and validate the physical device, the digital twin model must be refined. This paper describes the additional modeling levels required to perform such design. The additional models will allow designing and validating both the hardware components and the embedded software them. The main expected benefits are twofold: first, the elimination of the main bugs impacting the behavior and the performances of the end devices (e.g., power consumption, reach) and the whole system; second, the increase of the system robustness to facilitate and accelerate the complete system deployment in a more secure manner.

V. BRAIN-IoT MODELLING & VALIDATION FRAMEWORK

This section presents the BRAIN-IoT modelling & Validation framework developed to implement the methodology described in Section IV. BRIAN-IoT Modelling & Validation Framework defines a domain-specific Modelling Language describing IoT devices capabilities and system-level behaviors; it also provides toolset supporting the syntax of the modelling

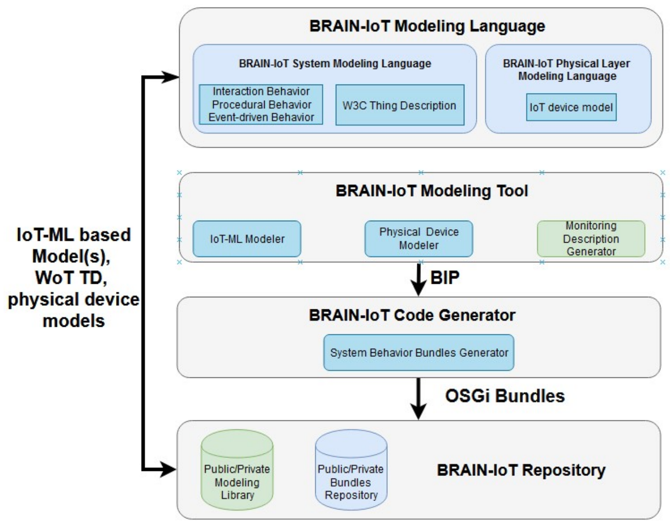

BRAIN-IoT Modeling Language
BRAIN-IoT System Modeling Language
Interaction Behavior
Procedural Behavior
Event-driven Behavior
W3C Thing Description
BRAIN-IoT Physical Layer Modeling Language
IoT device model
BRAIN-IoT Modeling Tool
IoT-ML Modeler
Physical Device Modeler
Monitoring Description Generator
BIP
BRAIN-IoT Code Generator
System Behavior Bundles Generator
OSGi Bundles
Public/Private Modeling Library
Public/Private Bundles Repository
BRAIN-IoT Repository
IoT-ML based Model(s), WoT TD, physical device models

Fig. 1. BRAIN-IoT Modelling & Validation Framework Components

language, allowing model verification, model checking, automatic code generation to provide rapid model-based development approach, and Models@Runtime monitoring features. The components in the BRAIN-IoT modeling & Validation framework are shown as in Fig. 1.

There are four main components in the BRAIN-IoT Modelling & Validation Framework: BRAIN-IoT Modelling Languages and Modelling Tools, BRAIN-IoT Code Generators, and BRAIN-IoT Repository. The detailed introduction for each component are presented in the following subsections.

A. *BRAIN-IoT Modelling Language*

The BRAIN-IoT Modelling Language is decomposed for three purposes: the system modelling language, the software modelling language, and the physical layer modelling language. Each of these modelling languages suits a different concern according to the abstraction layer. As a reminder, these abstraction layers are the system behavior, the software, and the physical layer. The authors of this paper believe using a domain-specific language, and de-facto common practice, suits the needs and habits of the domain-specific developer. Relationships are manually input to relate elements of each abstraction layer. This allows to have a holistic view of the whole architecture. Each of the following sub-sections describe a particular modelling language for a particular abstraction layer.

1) *System Modelling Languages and Tools:*

a) System Modelling Language: The IoT-ML is the system modelling language of BRAIN-IoT. It federates the specifications of heterogeneous sub-systems within a global IoT system. The language provides the necessary constructs to design the structural and behavioural system architecture of the system, entities abstracting the software, and entities abstracting the execution platform. At its conceptual core, IoT-ML integrates the concepts present in the IoT-A architecture reference model. Such a model is shown in Fig.2, extracted from the standard.

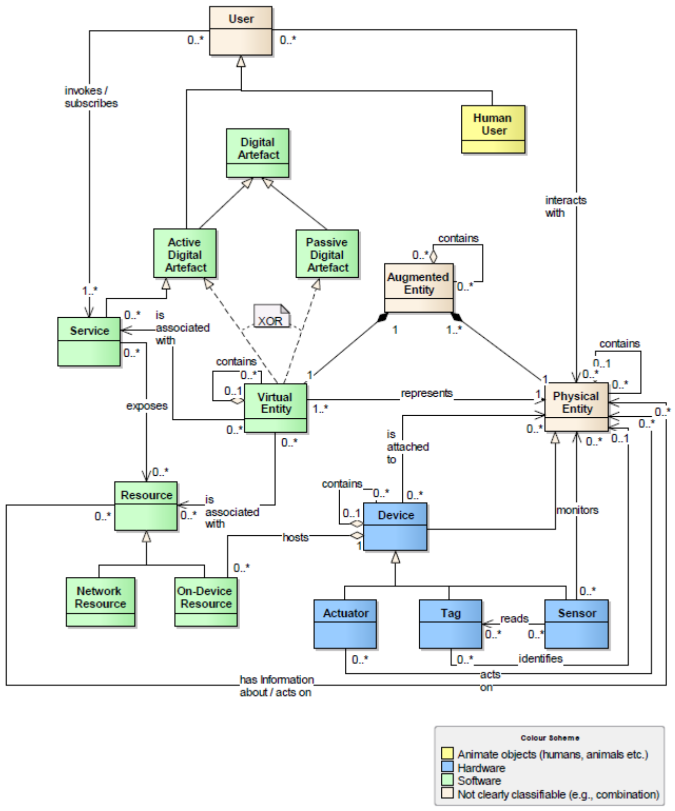


Fig. 2. IoT-A concepts in IoT-ML

IoT-ML is implemented as a UML profile. The UML is a generic modelling language with heavy roots in the object-oriented community. A UML profile is an extension of UML. It is composed of stereotypes that give additional syntax and semantics to the base UML elements that the stereotype extends. IoT-ML aggregates syntax and semantics from standard UML profiles to benefit from the languages they implement. SysML is used to benefit from its ability to describe and trace requirements. MARTE is used not only for the design of real-time embedded systems design, but also because it fosters the construction of models that may be used to make quantitative predictions taking into account IoT characteristics. IoT-ML takes a subset of stereotypes from such standards, and adds new stereotypes of its own representing concepts of IoT-A that are not present in MARTE or SysML. For example Virtual Entity, that can be both software and hardware, is a concept that does not exist in the UML standard profiles.

As a UML-based model, IoT-ML is a graphical modelling language. The language focuses on structural modelling. Such models are represented in UML composite structure diagrams. Components have internal structures that show parts exposing their interfaces through ports that are then connected together.

IoT-ML also focuses on behavioral modelling in the form of UML state-machines. In such models, the behavior represents the component's different states. States can pass from one to the other, based on captured events. Events may be due to operation calls or signalling. Specific behaviors may be executed upon transitioning across states, or entering / exiting / staying in a state.

While IoT-ML can already be used for domain-generic IoT systems modeling, within BRAIN-IoT, the authors of this paper have showcased that the core of IoT-ML (based on MARTE, SysML, extended with IoT-A concepts) is generic and rich enough to build domain-specific extensions for both IoT standards and IoT technologies. Indeed, IoT-ML has been extended with new concepts for the W3C TD standard. For example, the main concept of the W3C TD is the Thing, which can be either software or hardware or both. To showcase IoT-ML for a particular IoT technology, the authors of this work chose to integrate sensiNact [14] concepts into IoT-ML. The sensiNact platform interoperates several different middleware and communication protocols common to the IoT domain, e.g., Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP). Its particularity is that it has a common data model to represent all devices connected to different protocols. It is then possible monitor such devices' variables through common sensiNact API. The common API are also used in behaviors, described with the sensiNact domain-specific textual language, to prototype behaviors actuating the devices according to monitored variables.

Thanks to the profile mechanism of UML, all these domain-specific stereotypes can co-exist with the core IoT-ML stereotypes on the same UML base elements. Otherwise said, a same structural or behavioral element, of an IoT-ML architecture model, can be annotated with information specific to W3C TD or sensiNact. This fosters model re-use, separation of concerns, and model consistency through annotating the same base elements.

The mission of W3C WoT (Web of Things) is to counter the fragmentation in the IoT world through standardized complementing building blocks - e.g., metadata and Application Programming Interfaces (APIs) - based on Web technology. WoT enables easy integration and interoperability across IoT platforms and application domains. Therefore, the goal of WoT is to preserve and complement existing IoT standards and solutions like for the BRAIN-IoT domains Robotics and Critical Water Infrastructure. In this context, the usage of WoT-compliant *Thing Descriptions (TDs)* lays the foundation of interoperable standardized solutions for the various BRAIN-IoT domains and avoids their silo-like separation in order to overcome the problematic diversity of IoT systems.

There are several prominent W3C standard recommendations for WoT based on the W3C WoT architecture[1]; the WoT Architecture specification describes the abstract architecture for the W3C WoT. This abstract architecture is based on a set of requirements that were derived from use cases for multiple application domains. A set of modular building blocks is also identified whose detailed specifications are given in other documents. The architecture document describes how these building blocks are related and work together. Systems based on WoT architecture may cross different domains and integrate several vocabularies and ontologies.

[1]https://w3c.github.io/wot-architecture/

The WoT Thing Description (TD)[2] can be considered as the entry point of a Thing (much like the index.html of a Web site). Its specification is the core enabling technology. Different application layer protocols and media types can be described in a TD .

A TD abstracts the capabilities of individual Things into 3 categories called *Interaction Affordances*: *Properties* for sensing and controlling parameters, *Actions* for invocation of physical (and hence time-consuming) processes, and *Events* for the push model of asynchronous communication. A TD includes information models representing functions, transport protocol description for operating on information models, security information and general metadata about the device.

In summary, a WoT TD comprises the application logic requirements (e.g., values and alerts of a Thing). Devices are required to put a TD either inside them or at locations external to the devices, and to make the TD accessible so that other components can find and access them. As soon as available for a Thing, its TD can be used for flexible implementation and simulation (if required). To support the implementation, *WoT Scripting API* [3] specifies a common programming interface for Thing implementations as well as Consumer applications implemented by different programming languages.

b) System Modelling Tools and Relationship with Runtime: While IoT-ML is expressive enough to encompass IoT design, only with its modelling tools can exploit the full benefits of this formalism. One of the main goals of the IoT-ML modelling tools is to help deployment and connect it to deployed devices at runtime. This is accomplished through model transformation. Since IoT-ML, in BRAIN-IoT, has extensions specific to other IoT standards and technologies, the modelling tool offers transformation tools to go from one formalism to the other. The goal of such transformations is to bridge the gap between the runtime and the system model.

The IoT-ML models are made in the Papyrus modeller tool. The modelling tool is a typical Eclipse Eclipse Modeling Framework (EMF) [15] environment. It offers graphical editors, palettes to populate diagrams, and tree views to visit the hierarchical UML-based models.

An IoT-ML model, with TD stereotypes annotating its base elements, can be transformed to a TD in the JSON-based Serialization for Linked Data (JSON-LD) physical format. As a reminder, the TD files in JSON-LD are embedded on the real devices and polled at runtime. The authors of this paper believe this accelerates deployment of interfaces described in the system model. Furthermore, it bridges the gap between a natural way of describing architecture by the system engineers, and the text-based interface description by the developer. The importance to foster such a collaboration is explained in [16].

Using the same shared architecture model, the authors of this work can also transform the structure of the architecture

[2]https://w3c.github.io/wot-thing-description/
[3]https://w3c.github.io/wot-scripting-api/

into a sensiNact data model. The behaviors in the architecture, represented as state-machines, are transformed to equivalent sensiNact domain-specific language scripts. By then connecting to the sensiNact gateway, the data model and its sensiNact scripts can be run to monitor devices variables, and prototype system-level behaviors w.r.t. the runtime devices that should comply to the system model.

One last feature of the IoT-ML modelling tool is its ability to monitor its state machines. Although the connection between IoT-ML and sensiNact allows us to monitor variables and actuate devices, and therefore validate that the runtime is consistent with the system model, it is not always sufficient to understand the internal behaviors of the devices. For example, actuating a device, and noticing a variable change, may not be sufficient to understand what's happening for the human being. Therefore to provide human-friendly behavior explanations, it is possible to monitor state machines in an IoT-ML model. The state machines are animated and they mirror what's happening in the device state machines. What triggers the animations are messages that are sent to the IoT-ML modelling tool. Such messages are either sent by automatically generated code instrumentation points (i.e., during code generation itself of the state machine), or by any source that builds a string respecting the message format of the state machine monitoring tool.

The system model in IoT-ML, although connectable to existing runtimes, is not sufficient to develop the actual blocks that are to be deployed to form the runtime. Therefore its entities that represent software and hardware, must be refined, respectively, into a software architecture model and a physical architecture model. The next sections describe such models.

2) Physical Layer Modelling Approach: The BRAIN-IoT Physical Layer Modeling Approach allows the refinement of the digital twin model to an architectural model of the physical design including its functional and extra-functional properties. This model can be directly used by the device as well as the Integrated Circuits (ICs) designers as a reference model. This proposal follows a top-down model-based design approach composed of black box and white box models, called virtual twins. They are functionally equivalent to the physical IoT devices and can be used to serve different purposes. One model can be seamlessly replaced by the other, as they all share the same functional specification that represents faithfully the IoT device at its boundaries. They feature the functional and extra-functional properties of the IoT device (behavior, security, energy efficiency, reach, etc.).

a) Black box model: The black box (BB) model is an abstract representation of the IoT device. It is a simple service-oriented model that represents its functionality, regardless of the internal architecture that implements its behavior. The BB model can be considered as the functional reference of the end-device. It can be reused whatever the implementation choices, or even in case of replacement of the physical device by another one, as long as the functional specification and interfaces remain unchanged. It provides the functional contract, the other models or the physical device shall comply to. Executable, it is a non-ambiguous, repeatable and deterministic model of the end-device specifications. It abstracts the internal architecture of the end-device, as well as the embedded software. The BB model takes as input a file containing the data values that would be obtained from a real sensor operating in real conditions.

b) White box model: The second step of the top-down methodology is the creation of a white box (WB) model of the end-device representing the internal architecture of the device.

This architecture is composed of one or several sensors or actuators, a micro-controller, and one or several connectivity elements. Sensors are typically exposing an Inter-integrated-circuit (I2C) interface for digital data (or I/Os for analogue one), to let the microcontroller (MCU) read and write into registers to gather the data from the sensor, while the connectivity Internet Protocols (IPs) can be programmed through a Serial Peripheral interface (SPI) bus or through a Universal Asynchronous Reception and Transmission (UART) connection. The Hardware Abstraction Layer (HAL) of the MCU provides an API to access the hardware resources from the embedded software.

The White-box model represents this typical architecture that is described using the SystemC/TLM IEEE 1666 modeling language [17]. The model of the micro-controller typically includes a model of the embedded processor, such as Quick EMUlator (QEMU) or Instruction Set Simulators, and the models of all the peripheral blocks. This list obviously varies with each micro-controller, but usually includes the timers, the reset / clock / power controllers, the interrupt controller, and the hardware accelerators available for the part number. It also includes I/O models - UART, GPIO, I2C or SPI controllers - that are used to interact with the other elements of the end-device. The MCU is modelled to accurately represent the bus transactions initiated by the processor. The sensor model serves I2C requests issued by the micro-controller, and implements the behavior of the block, to react to the programming sequences. The connectivity model serves SPI or UART requests issued by the micro-controller, and implements the behavior of the block, to react to the programming sequences. In the current developments, the authors have decided to perform the communication as an abstraction of all the communication data path by issuing Hypertext Transfer Protocol (HTTP) requests to the network. When detailed communication protocol information is needed, the connectivity models can be connected to an elaborated communication model including communication medium such as LoRaWAN, serving protocol-specific commands. The WB model conforms to the functional contract of the end-device, as prescribed by the BB model.

c) Benefits of the Physical Layer Modelling Approach: The benefits BRAIN-IoT physical modeling language brings to the IoT domain are 1) Early verification of the embedded software, in charge of: data gathering from sensors; local processing (data formatting, data analysis, power management, payload construction, encryption, etc.); transmission of the encrypted data or metadata using the device connectivity capabilities (Bluetooth, LoRa, SigFox, etc.). 2) A system verification can be achieved in advance without the debug limitations inherent to physical devices, the models offer full inspection and observabilicould yty capabilities for debug and analysis; 3) The complexity of large-scale systems (from tens to thousands of end-devices) can be addressed by instantiating the appropriate number of models in the simulation platform; 4) The system reliability is increased, as the validation strategy of the end-device is strengthened by adding scenarios focusing on device robustness considering its interaction with system environment.

3) Formal Software Modelling Language and BRAIN-IoT Code Generator: The Behaviour, Interaction, Priority (BIP) Modeling language, introduced in [18], is the software modeling language in BRAIN-IoT. It supports the methodology for building systems from atomic components. It uses connectors, to specify possible interactions between components, and priorities, to select amongst possible interactions.

BIP is a highly expressive component-based language that supports the specification of composite, hierarchically structured components starting from the atomic ones. In BIP, the atomic components are finite-state automata having transitions labeled with ports and states that denote control locations where component waits for interactions. Ports are actions that can be associated with data stored in local variables and used for interactions with other components. Connectors relate ports from components by assigning them to a synchronization attribute, which may be either trigger or synchronous. A compound type defines a level of the hierarchy. It contains instances of component and connectors types (i.e., subcomponents) with connection definitions and also priorities to schedule the interactions between these components. A compound component offers the same interface as an atom, so, externally, there is no difference between a compound and an atom. Inner ports from sub-components can be exported.

The BIP formalisms allow the rigorous specification and analysis of IoT systems components behavior. Moreover, the component-based approach supported by BIP facilitates portraying behavior with reusability, and maintainability features.

The BRAIN-IoT Code Generator relies on BIP language to describe the system behavior. It accepts as an input a formal specification of system architecture and system behavior using the BIP language, then it translates the system model into a set Java code artifacts. The generated Vanilla code could be simulated independently to the BRAIN-IoT execution platform called Fabric [19], and thus, the user could check the validity of the behavior specified at the BIP level. When the code is simulated and validated by designers, the code is wrapped in an envelope called bundles that fit the Brain-IoT execution platform. BRAIN-IoT Code Generator is an Eclipse plugin that includes two modules: (i) BIP language processor with the full support of BIP grammar and syntax checking, (ii) Java code generator of BIP model.

Using BRAIN-IoT Code Generator, the mapping of a formal BIP language integrates all the structures related to components development engineering such as composability and reusability. Moreover, the target language is independent of any technology; all the existing operating systems embed the processing ability of JAVA language.

The information flow of the components described above within the BRAIN-IoT modeling & validation framework is shown in Fig. 1. It is firstly responsible for designing IoT system application, which is going to be constructed, based on the actions and relations between the available devices - e.g., sensors, actuators, Cyber Physical Systems (CPSs) - and external services, e.g., weather forecast, open data, third-party IoT platforms, databases. The system level application is modelled along with the relevant IoT environment using BRAIN-IoT System Modeling Language (IoT-ML) through BRAIN-IoT modeling tool, representing the system-level behavior models and describing its self-adaptive behavior. Then, the IoT-ML model is refined to BIP model representing functional software-level components model. Finally, the BIP model is converted in source code as system behavior OSGi bundles through BRAIN-IoT Code Generator. The generated bundles are then released and stored in BRAIN-IoT repository, to be deployed and executed in the production environment. Furthermore, it also offers the ability to use development-time models to supervise running execution platform states. This solution enables monitoring the IoT devices' status and system configurations. This is possible, because BRAIN-IoT modeling tool supports the Models@runtime paradigm, allowing to synchronize the system's behavior and the real system. In addition, the generated OSGi bundles can be validated leveraging IoT device models developed with BRAIN-IoT physical layer modeling language to validate the correctness of the system behavior, before deploying in the physical world. In the BRAIN-IoT repository, stores the public/private modeling library that contains the system level models, BIP behavior models, and WoT TD for the IoT device/platform. In the BRAIN-IoT Service Artifacts library, there are System Behavior Artefacts generated from the BRAIN-IoT Code Generator.

VI. BRAIN-IoT Use Case

In this section, the methodology and the modeling framework proposed has been evaluated using BRAIN-IoT use cases [20].

A. Service Robotic

In a warehouse there are two zones: a loading area and a storage area. These zones are divided by an automatic door. This door has a QR code attached on each side which is legible only when the door is closed. In the loading area there are 3 carts each with a QR code attached (different for each case) and three robots responsible for warehouse management are also located. The robots are equipped with vision cameras that allow reading the QR codes they find. The model of the robots is a *RB1* base from Robotnik company. They are capable of detecting and raising carts and also transport them from one point to another in an autonomous way. The warehouse is controlled through an Orchestrator that assigns the carts to the storage areas and orchestrates the robots. On the other hand, the storage area is divided into 3 sub-zones (A, B, C) where

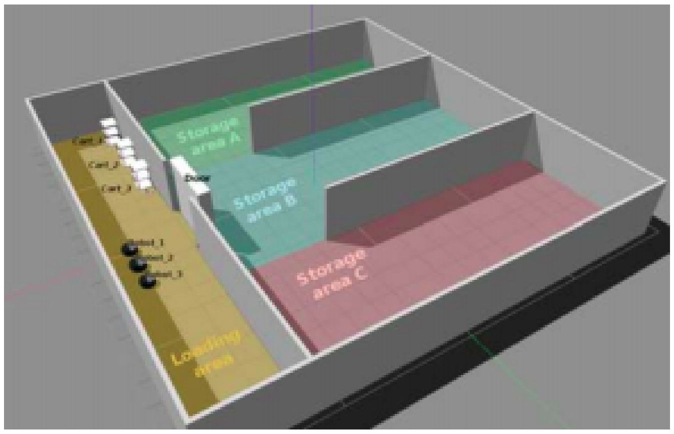


Fig. 3. Simulation environment of the Use Case

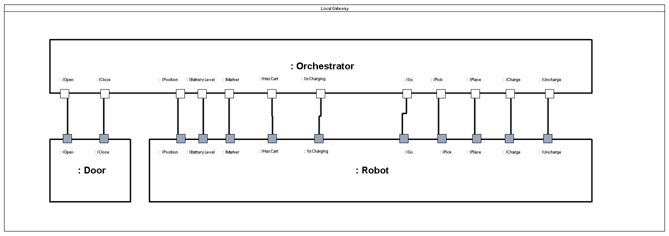
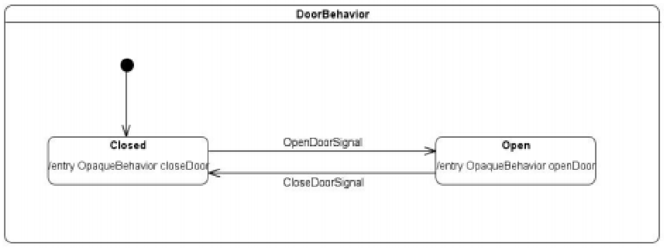


Fig. 4. Example of Composite Structure Diagram in IoT-ML Describing Robot, Door, and Orchestrator

carts can be stored. The Fig. 3 shows the scenario of the use case in the simulation environment. The logic of the system is that the robot takes the cart from the loading area and places it in the storage area, on the way, it detects an elevator door. The robot scans the QR code of the door. It then sends the QR code to a Fleet Management System (FMS) with a door open request, FMS opens the door. After the door opened, the robot will send again the QR code to the FMS with a door close request.

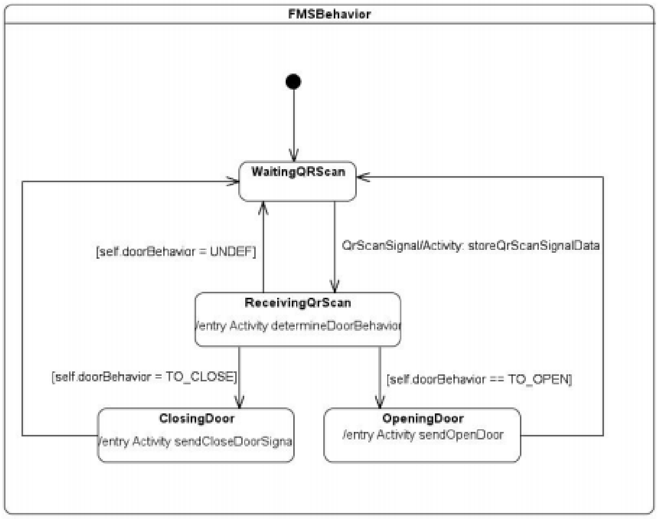
Firstly, the authors modeled the system level components and the interaction interfaces using IoT-ML with the BRAIN-IoT as shown in Fig. 4, in which, the FMS is represented by an Orchestrator, with a robot and door connected to an Orchestrator that sends commands to both devices. Moreover, the behavior of the FMS is modeled using BRAIN-IoT modeling language with BRAIN-IoT modeling tool as shown in Fig. 6, and the behavior of the door in Fig. 5

Then the system models are refined as BIP software models and as the inputs of the code generator, the generated artefacts are deployed in the robot. While the robot is running, its status and warehouse coordinates configurations will be monitored through the monitoring tools. The software architecture is as shown in Fig. 7. As a reminder, the EventBus is a service provided in BRAIN-IoT for the communication in a distributed environment and the Robotic Operating System (ROS) Edge Node is an adaptor deployed in the robot for communicating with other applications deployed in the BRAIN-IoT Fabric through the EventBus. Orchestrator Behavior represents the



DoorBehavior
Closed
/entry OpaqueBehavior closeDoor
OpenDoorSignal
CloseDoorSignal
Open
/entry OpaqueBehavior openDoor

Fig. 5. Door Behavior



FMSBehavior
WaitingQRScan
[self.doorBehavior = UNDEF]
QrScanSignal/Activity: storeQrScanSignalData
ReceivingQrScan
/entry Activity determineDoorBehavior
[self.doorBehavior = TO_CLOSE]
[self.doorBehavior == TO_OPEN]
ClosingDoor
/entry Activity sendCloseDoorSignal
OpeningDoor
/entry Activity sendOpenDoor

Fig. 6. Fleeet Management System Behavior

system level behavior and orchestrates the robot and the door. ROS Edge Node provides the adaptor for ROS environment to communicate with other components via eventBus, the current version is extended with Behavior Translator to connect Papyrus using User Datagram Protocol (UDP). Papyrus will provide the visual realtime monitoring of the robot state transitions with the state machine, it is integrated with ROS Edge Node. To demonstrate the Models@Runtime feature, here the authors monitored two aspects: one is the configuration of the warehouse coordinates, another is the runtime status of the robot.

a) Runtime Robot Status Monitoring: The sequence of the workflow is as following: 1) ROS Edge Node receives the command events from Orchestrator. 2) ROS Edge Node sends the command to the robot, meanwhile, it converts the

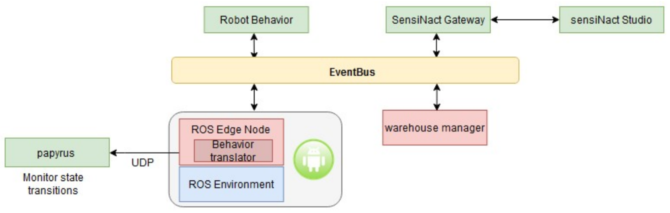


Fig. 7. Service Robotic Monitoring

command to the messages can be received by Papyrus. 3) When Papyrus receives the command transition message, it will dynamically display the state change.

b) Runtime Warehouse coordinates monitoring: Before the application is deployed, the end-user will configure the warehouse through SensiNact studio, these values will be delivered to warehouse manager through SensiNact Gateway and EventBus, then stored in three tables, which are picking points table storing the coordinates in the loading area, storage points table storing the coordinates in the storage area, and cartStorage points table storing the corresponding place point in the storage area of each cart. On the other side, during runtime, whenever a robot changes the attribute value in the table due to the mission of moving a cart, the warehouse manager will send an update event to sensiNact Gateway, then be delivered to sensiNact studio. Hence, the updates will be reflected in the sensiNact studio.

B. Critical Water infrastructure management

The services of water supply are associated to a series of infrastructures that are considered, in accordance with European norms, as critical, and therefore, they are bound to a series of conditions for their development, especially in the technological aspect.

In the water management sector, most of the processes are associated to disperse infrastructures in large and varied geographical sites, with numerous interactions with other elements and services related to the human activities. The sharing of information in a safe and efficient manner is a challenge to optimize the actions in urban surroundings to simplify and improve the citizen's life. The development of models and their implementation in multi-access platforms (internal usage, clients, responsible entities, etc.), constitutes a knowledge and technological challenge for the sector. They require development for the correlation of data, its analysis and the creation of indicators and processes for a better usage of the infrastructure's maximum potential.

In the water management use case, the scenarios aim to leverage prediction models (based on the collected data), to: increase the security of water supplies, optimize the underlying costs, enhance the services for end-users and connect the infrastructure to other urban services. Analyzing gathered data will help to create more accurate indicators for decision-making, and for real-time, smart and adaptive control procedure and generally more efficient and automated business processes. For evaluating scenarios and developments carried on during the project, a mock-up called MEDUSA was built in the facilities. In the Critical Infrastructure of Water Management System, four use cases have been defined to evaluate the main features, concepts and developments of the BRAIN-IoT project. In this paper for space reasons, only the Resilience Use Case is described. The objective of this Use Case is to validate the resilience of the system in case of hydraulic failures. Normally, the hydraulic system works according to a defined consumption curves at the entrance of the water meters sections. These curves represent the real

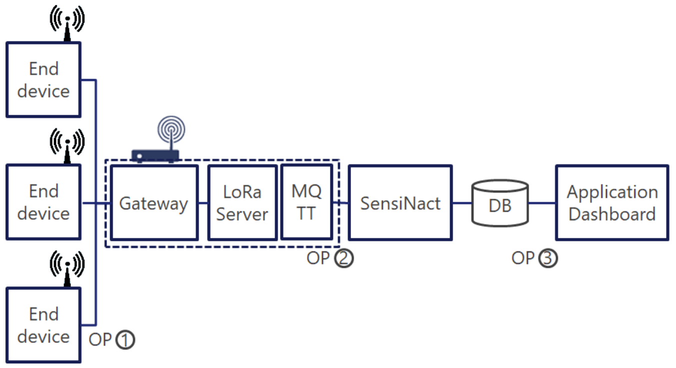


Fig. 8. Model of critical water infrastructure

consumption in various points of Coruña city (Elevado, Cola and Cabecera). The curves are obtained with the opening and closure of the electric valves of every section of water meter, simulating the real consumption of the customers. The resilience of the system is validated in terms of guarantee that the consumption can be ensured in those points. One scenario is to simulate the failure of an electric valve in a pipe and to ensure that BRAIN-IoT platform can recirculate the water for another pipe, controlling the electric valves that allows that, and according to the Detection and Responsive System (DRS) that have been trained for these real failures. Another possible scenario is to simulate the failure of a flow meter and to ensure that BRAIN-IoT platform can recirculate the water for another section with the same goal of ensuring the consumption curves. DRS uses Machine Learning techniques for the detection of failures. The responsive part is based on defined rules, acting as expert system. In the scenario described above, there are three main parts involved: 1) data gathering from the IoT devices 2) An algorithm for detecting the anomalies/failure 3) Control system for reacting the abnormal situation. Point 2 and 3 will be the main components of DRS.

In the critical water infrastructure architecture (see Fig. 8), heterogeneous sensors are placed in various remote locations to collect and transmit data through a LoRaWAN network, obtaining a huge amount of data to be used for training anomalies detection algorithm with the limited number of physical devices. The authors have developed the simulated IoT devices models to generate the data. The system model performs data gathering (from an input table), data encapsulation and encryption, and data transmission to the network through HTTP requests, as an abstraction of the complete LoRaWAN communication data path: gateway + LoRa server / MQTT. Then, the data are processed and stored in the edge database. System security and robustness against vulnerabilities and attacks are guaranteed.

Fig. 9 shows an example of the simulation results, at the system level, obtained by the data transmission, data recovery, processing and display in the EMALCSA application dashboard of a water meter end device. The curves represent, for each device: i) the percentage of valve openness, ii) the water flow measured as output of the valve, iii) the saturation

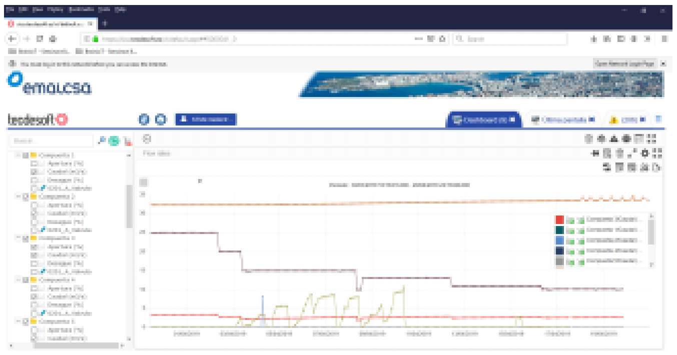


Fig. 9. Plot, in end user dashboard, of the data gathered by a water meter end device model

of the pipe.

The authors also aim implementing the control system in point 3 using the BRAIN-IoT Modeling & Validation Framework, following the proposed modeling methodology to get the intended system control models, then, generate the application artefacts. Its correctness will be validated in the simulated devices, hence the risk of physical critical infrastructure damage could be reduced before deploying to the real environment.

VII. Conclusion and Future Works

This paper has presented the Model Based Methodology and Framework proposed for design and Management of next-gen IoT Systems. This solution provides a Model-Based Engineering (MBE) approach to ease the development of the IoT systems. It offers a system-level model, which captures the system functionalities and behaviors to help refinement of the software-layer modelling; it facilitates the linking towards real devices and external services through meta-data representation in WoT TD; the application code is generated from model for monitoring and controlling the IoT infrastructure; it supports the system application validation leveraging the simulated IoT devices developed with the BRAIN-IoT physical layer modeling language; finally, it allows monitoring the IoT system behaviours and its configurations in a human-friendly graphical manner through the Models@Runtime approach at the execution time.

As future work, the Modelling & Validation framework will be extended to also support AI modelling. In particular the authors will evaluate the feasibility of using system modeling languages to either describe finely machine learning algorithms, or their deployment, in the goal of accelerating development and promoting interoperability between AI (sub)-systems. Furthermore, the Critical Water Infrastructure management use case will be further developed including the control part and the associated actuators models to demonstrate the complete functionalities provided by the modeling framework. Furthermore, the authors would provide a survey to some users and get some evaluations to demonstrate the benefits brought by the solutions proposed in this paper.

Acknowledgment

The work presented here was part of the project "Brain-IoT-model-Based fRamework for dependable sensing and Actuation in iNtelligent decentralized IoT systems" and received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780089. The authors thank Robotnik Automation S.L.L. and EMALCSA for providing the environment, real and simulated to test the solution.

References

[1] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[2] O. Vermesan and J. Bacquet, *Next generation Internet of Things: Distributed intelligence at the edge and human machine-to-machine cooperation*. River Publishers, 2019.

[3] AGeSys Consortium. AGeSyS Atelier de Genie Systeme. [Online]. Available: https://www.aerospace-valley.com/sites/default/files/encart_html/index.html

[4] Y. Sun, G. Memmi, and S. Vignes, "A model-based testing process for enhancing structural coverage in functional testing," in *Complex Systems Design & Management Asia*, 2016.

[5] S. Dhouib, A. Cuccuru, F. Le Fèvre, S. Li, B. Maggi, I. Paez, A. Radermarcher, N. Rapin, J. Tatibouet, P. Tessier *et al.*, "Papyrus for iot—a modeling solution for iot," *Proceedings l'Internet des Objets (IDO: Nouveaux Défis de l'Internet des Objets: Interaction Homme-Machine et Facteurs Humains. Paris, France*, 2016.

[6] P. Roques, "Mbse with the arcadia method and the capella tool," in *Proceedings of ERTS 2016*, Toulouse, France, 2016.

[7] Z. M. Bzymek, M. Nunez, M. Li, and S. Powers, "Simulation of a machining sequence using delmia/quest software," *Computer-Aided Design and Applications*, vol. 5, no. 1-4, pp. 401–411, 2008.

[8] M. Bauer, M. Boussard, N. Bui, F. Carrez, C. (SIEMENS, J. (ALUBE, C. (SAP, S. Meissner, A. IML, A. Olivereau, M. (SAP, W. Joachim, J. Stefa, and A. Salinas, "Internet of things – architecture iot-a deliverable d1.5 – final architectural reference model for the iot v3.0," 2013.

[9] H. Espinoza, D. Cancila, S. Gérard, and B. Selic, "Using marte and sysml for modeling real-time embedded systems," *Model-Driven Engineering for Distributed Real-Time Systems: MARTE Modeling, Model Transformations and their Usages*, pp. 105–137, 2013.

[10] W. Emmerich and N. Kaveh, "Component technologies: Java beans, com, corba, rmi, ejb and the corba component model," in *Proceedings of the 8th European software engineering conference*, 2001.

[11] Object Management Group, "Dds for lightweight ccm (dds4ccm)," 2009.

[12] B. Elvesæter, C. Carrez, P. Mohagheghi, A.-J. Berre, S. G. Johnsen, and A. Solberg, "Model-driven service engineering with soaml," in *Service Engineering*, 2011, pp. 25–54.

[13] Object Management Group, "Unified component model for distributed, real-time and embedded systems," 2020.

[14] L. Gürgen, C. Munilla, R. Druilhe, E. Gandrille, and J. Nascimento, *sensiNact IoT Platform as a Service*, 08 2016, pp. 127–147.

[15] F. Budinsky, "The eclipse modeling framework," *Doctor Dobbs Journal*, vol. 30, pp. 28–32, 08 2005.

[16] V. C. Pham, S. Li, A. Radermacher, S. Gérard, and C. Mraidha, "Fostering software architect and programmer collaboration," in *Proceedings of the 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2016.

[17] "IEEE 1666-2011; IEEE standard for standard systemc language reference manual," standard, 2011.

[18] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the BIP framework," *IEEE Software*, vol. 28, no. 3, pp. 41–48, May 2011.

[19] R.Nicholson, T.Ward, D.Baum, X.Tao, D.Conzon, and E.Ferrera, "Dynamic fog computing platform for event-driven deployment and orchestration of distributed internet of things applications," pp. 239–246, 2019.

[20] E. Ferrera., X. Tao., D. Conzon., V. S. Pombo., M. Cantero., T. Ward., I. Bosi., and M. Sandretto., "Brain-iot: Paving the way for next-generation internet of things," in *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*, INSTICC. SciTePress, 2020, pp. 470–477.