UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MATHEUS TAVARES FRIGO

Process-Oriented Approach for configuring IoT environments and applications

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Lucineia Heloisa Thom
Coadvisor: Dr. Pascal Hirmer

Porto Alegre
November 2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões Mendes
Vice-Reitora: Prof^a. Patrícia Helena Lucas Pranke
Pró-Reitora de Ensino: Prof. Cíntia Inês Boll
Diretora do Instituto de Informática: Prof^a. Carla Maria Dal Sasso Freitas
Diretora da Escola de Engenharia: Prof^a. Carla Schwengber Ten Caten
Coordenador do Curso de Engenharia de Computação: Prof. Walter Fetter Lages
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro
Bibliotecária-Chefe da Escola de Engenharia: Rosane Beatriz Allegretti Borges

“The great growling engine of change - technology.”

— ALVIN TOFFLER

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family for all the support over the past six years. Especial thanks to my father, mother, and grandmother, who were present from the beginning of my computer engineer journey. I would also like to express my immense feelings of gratitude to my friends, especially Luís and Nicholas - who walked side by side with me until the end. I also thank all the other friends who were with me at some point in these six years, supporting me and creating great memories and memorable experiences. Finally, I would also like to thank Amanda for her support, especially during the writing of this work.

My deepest gratitude to the computer science institute at UFRGS for providing such an enriching environment. And especially thanks for the opportunity given to me by the institute to do the exchange in Germany, whose experience was essential for me to become who I am today.

I am also profoundly grateful for all the help my advisor, Professor Lucineia Heloisa Thom, provided with articles, life advice, or other moments and teachings. Thank you for believing in me from the moment we met there in Germany. My sincere thanks also to my co-advisor Pascal Hirmer, and also to Ana Cristina Franco da Silva, who, in addition to teaching me a lot about the Internet of Things area, made the academic experience of my exchange program the best possible.

ABSTRACT

The Internet of Things (IoT) is a paradigm that consists of heterogeneous devices with capabilities of sensing, acting, processing, communication, networking, and storage. These devices are highly interconnected to reach a common goal – making people's life easier. In Smart Factories, for example, highly automated production processes can be enabled leading to more efficiency and reduced costs. In the IoT, communication and continuous transmission of collected data are conducted through standardized Internet protocols. Well-known applications of the IoT include Smart Homes, Smart Factories, or Smart Cities. Nevertheless, setting up new IoT environments usually requires manually setting up the devices, configuring sensors, or writing scripts. Conducting these complex steps can be very cumbersome, especially finding suitable technologies, protocols, or standards. The heterogeneity of smart devices, communication protocols, and other technologies and the high distribution in the IoT makes the configuration of smart environments and applications even more complex and time-consuming. Therefore, in this work, we propose a holistic method to help domain experts setting up IoT environments and applications. Our method is composed of (i) a toolbox with common building blocks of the IoT, and (ii) a business process-based approach to orchestrating the setup of these building blocks. Through our approach, domain experts can select the building blocks they require for their IoT application and generate a step-by-step manual for their setup.

Keywords: Internet of Things. Business Process Management. Process-oriented approach. Toolbox. Holisthic method.

Abordagem orientada a processos para configuração de ambientes e aplicações IoT

RESUMO

A Internet das Coisas (IoT) é um paradigma que consiste de dispositivos heterogêneos com capacidades de detecção, atuação, processamento, comunicação, armazenamento e de rede. Esses dispositivos são altamente interconectados para atingir um objetivo em comum – tornar a vida das pessoas mais fácil. Em indústrias inteligentes, por exemplo, processos de produção intensamente automatizados podem ser estimulados levando a mais eficiência e custos reduzidos. Na IoT, a comunicação e a transmissão contínua de dados coletados são conduzidas através de protocolos de internet padronizados. Aplicações bem conhecidas de IoT são Casas Inteligentes, Indústrias Inteligentes, e Cidades Inteligentes. Contudo, a configuração de novos ambientes inteligentes comumente requer a montagem manual dos dispositivos, configuração dos sensores ou escrita de scripts. A realização dessas etapas complexas pode ser um trabalho árduo, especialmente no que diz respeito a encontrar tecnologias, protocolos ou padrões adequados. A heterogeneidade de dispositivos inteligentes, protocolos de comunicação e outras tecnologias, além da alta distribuição na Internet das Coisas torna a configuração de ambientes inteligentes ainda mais complexa e demorada. Desse modo, nesse trabalho, propomos um método holístico para auxiliar na configuração de ambientes e aplicativos IoT. Nossa abordagem é composta por (i) uma caixa de ferramentas com blocos de construção comuns da IoT, servindo de base para nosso método, e (ii) uma abordagem orientada a processos de negócio para orquestrar a configuração dos blocos de construção. Por meio da nossa abordagem, os especialistas do domínio podem selecionar os blocos de construção necessários para a sua aplicação IoT e também gerar um manual passo a passo para sua configuração.

Palavras-chave: Gerenciamento de Processos de Negócio. Internet das Coisas. Abordagem orientada a processos. Caixa de Ferramentas. Método holístico..

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management Systems
CBSE	Component-Based Software Engineering
CRUD	Create-Read-Update-Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText markup Language
IOT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NPM	Node Package Manager
REST	Representational State Transfer
RPC	Remote Procedure Call
SPA	Single Page Application
UI	User Interface

LIST OF SYMBOLS

$\neq$	Not equal
$\emptyset$	Empty set
$\in$	In
$\rightarrow$	To
$:=$	Equal

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

The Internet of Things (IoT) is a technological paradigm where a large number of heterogeneous interconnected devices communicate through standardized Internet protocols to share information or to perform actions (VERMESAN; FRIESS, 2013). The IoT results from the advancement in many parallel and often overlapping fields. Some of the fields that impact IoT expansion are embedded systems, ubiquitous and pervasive computing, mobile telephony, telemetry, machine-to-machine communication, wireless sensor networks, mobile computing, and computer networking (TEKINERDOGAN; KOKSAL, 2018). Usually, devices of the IoT are embedded systems with electrical or electronical components, software, sensors, actuators, and network connectivity, that enable sensing, acting, collecting, and exchanging data via the network (JANIESCH et al., 2017). Moreover, each device is uniquely identifiable through its embedded computing system and interoperates within the existing network infrastructure.

According to Cook and Das (2007) smart environments enable to acquire and apply knowledge about the surroundings and its inhabitants to improve their experience in that context. Therefore, the IoT enables the creation of new kinds of applications, such as Smart Homes (HARPER, 2006), Smart Cities (CHOURABI et al., 2012), or Smart Factories (SHROUF; ORDIERES; G.MIRAGLIOTTA, 2014).

The interconnection of these IoT devices is expected to help in automation in many fields as it enables continuous monitoring based on sensing devices and analytical opportunities in smart environments and the possibility to actuate feedback. According to Atzori et al. (2010) some of the fields impacted by the IoT are domotics (science that concerns electronics and information application to domestic life), assisted living, e-health, enhanced learning, automation and industrial manufacturing, logistics, business process management, and intelligent transportation of people and goods. Moreover, the interconnection of IoT devices also creates opportunities for more direct integration of the physical world into computer-based and digitized systems, improving efficiency, accuracy, and economic benefits besides increased automation and reduced human intervention (JANIESCH et al., 2017).

However, building IoT systems can be very complex (MCEWEN; CASSIMALLY, 2013) since the devices, protocols, technologies, and standards are highly heterogeneous. There is a plethora of devices able to perform a specific task with different levels of performance and costs. For example, Raspberry Pis are powerful devices able to provide an operating system and built-in communication technologies, such as WiFi or Bluetooth, allowing it to process, communicate, and store. In contrast, there are very restricted IoT devices, such as micro-controller boards, that do not provide any sophisticated operating system, sometimes just capable of simple data pre-processing and communication (e.g., through either WiFi, Bluetooth, or cable), providing only a limited runtime environment to run small scripts. Besides, there is a lack of IoT application development approaches that combine IoT application requirements for heterogeneity and analyze them according to the *quality of service* (BALEN; ŽAGAR; MARTINOVIC, 2011) requirements of IoT applications at an early stage of application development.

Furthermore, in the IoT, not only the devices are very heterogeneous but also the communication protocols. There are many alternative communication solutions with diverse performance characteristics, e.g., CoAP (Bormann; Castellani; Shelby, 2012), MQTT (Hunkeler; Truong; Stanford-Clark, 2008), LORA (AUGUSTIN et al., 2016), or HTTP. These numerous available solutions provide different features and performance trade-offs, making it very complex to select the most suitable IoT communication technology for a particular smart environment (GOMEZ et al., 2019).

Another challenge is related to building an optimal system for a domain-specific application. IoT systems are highly heterogeneous, and a computing unit in a smart environment can have dramatically varying computing power, storage capabilities, network bandwidth, and energy requirements. Other aspects, like I/O mechanisms, sensory capabilities, supported input modalities can also vary considerably (TAIVALSAARI; MIKKONEN, 2017). Not only a device specification but also its operation, for example, different sensors and actuators can operate in other communication protocols, and it can have a considerable impact on the application performance.

This abundance of devices, communication protocols, and technologies hampers the selection of the proper hardware components and protocols for building IoT environments and applications. Hence, there is a demand for coping with this considerable heterogeneity and to support IoT developers choosing the proper hardware, technologies, and protocols for their applications.

To cope with the complexity and to reduce the development time of building IoT systems, one goal of the present work is the development of a holistic method to support domain experts without necessary deep technical understanding in the whole process of setting up the desired IoT system, covering requirements elicitation, choosing suitable components, protocols, and technologies, and generating a step-by-step manual for the setup.

First, we introduce a toolbox containing common building blocks that are regularly used in the creation of IoT environments and applications. These building blocks can represent different parts of an IoT environment, e.g., hardware components, network protocols, message brokers, gateways, platforms, or other software components. A building block, in our context, consists of a high-level description to be understandable by domain experts as well as specific implementations. In our approach, the building blocks are provided by experts in building IoT environments and are included in our toolbox, which domain experts can access.

Second, we introduce a business process-based approach to set up the IoT environments based on the suggested building blocks. Usually, setting up IoT environments requires many different steps that can be conducted either in parallel or sequentially in case of any dependencies. For example, before connecting a device to an IoT platform, a network connection needs to be set up. Hence, to enable a robust setup of IoT systems, it makes sense to use Business Process Management (BPM) (DUMAS et al., 2018) for the orchestration of these steps.

BPM is a well-established discipline that deals with the identification, discovery, analysis, (re-)design, implementation, execution, monitoring, and evolution of business processes. A business process is a collection of inter-related events, activities, and decision points involving actors and resources and collectively lead to an outcome that is of value for an organization or a costumer (DUMAS et al., 2018). Moreover, the Business Process Model and Notation 2.0 (BPMN 2.0) (OMG, 2014) is a commonly used language for process modeling.

Several works are emerging in the literature combining BPM and IoT, e.g., utilizing sensor data to enable the actuation on services or adapting running business processes to align them with the state of the smart things continuously (e.g., assets, humans, and machines) (JANIESCH et al., 2017). Process analytics, execution, and monitoring based on IoT data can enable an even more comprehensive view of processes and realize the potential for process optimization. It is known that in many areas, such as supply chain management, intelligent transport systems, domotics, or remote healthcare, business processes can gain a competitive edge by using the information and functionalities provided by IoT devices (DOMINGOS; MARTINS, 2017).

The other goal of this work is to discuss the mutual benefits of both BPM and IoT working together, modeling and executing processes taking into account the IoT devices (i.e., IoT-aware processes), as business processes can use IoT information to incorporate real world data, to take informed decisions, optimize their execution, and adapt itself to context changes (JEDERMANN; LANG, 2008). For example, a smart factory might be better manageable when closely linking the digital process with the physical world as enabled by the integration of IoT and BPM. In this scenario, the completion of manual activities can be made observable through the usage of appropriate sensors. IoT can complete BPM with continuous data sensing and physical actuation for improved decision-making. Moreover, the increase in processing power of IoT devices enables them to become active participants by executing parts of the business logic. IoT devices can aggregate and filter data and make decisions locally by performing business logic functions whenever central control is not required, reducing both the amount of exchanged data and central processing (HALLER; KARNOUSKOS; SCHROTH, 2008).

1.1 Research Questions and Goals

The IoT has a pervasive impact on society, and an increasing number of systems are now based on IoT (HALLER; KARNOUSKOS; SCHROTH, 2008). The major challenge for our work is the significant heterogeneity of the IoT domain. This brings up many problems when dealing with IoT systems and all the plethora of heterogeneous smart devices, communication protocols, and other technologies. First, this wide variety introduces the challenge of creating and using a uniform way of describing things in IoT environments and applications regarding what the particular thing is, what it does, and how it communicates (KHALED et al., 2018). Based on these observations, the following research question is raised:

- **[RQ1]**: Is it possible to represent the considerable diversity of components of the IoT domain using Building Blocks?

In addition, the substantial growth of IoT environments and applications and the usage of smart devices have increased the complexity in managing such environments, especially the highly heterogeneous devices, due to their specific features and characteristics (e.g., sensitivity, communication, response time, resource constraint). Therefore, to leverage the potential supplied by these heterogeneous IoT devices, there is a visible need to bridge the gap between IoT and the BPM field. IoT-aware processes need to be efficiently designed in an unambiguous manner to achieve the association of IoT devices and their capabilities to activities in a process model.

However, it is essential to formally and explicitly define the characteristics of the IoT resources participating in a process, e.g., two devices measuring the same metric, such as the room temperature, may have non-trivial differences in their capabilities, such as sensitivity or response time (SURI et al., 2017). Further, these observations raised another research question:

- **[RQ2]**: Can IoT Building Blocks help in the IoT-aware process modeling?

The general goal of this work is to provide a holistic method that describes the necessary steps domain experts need to take to set up IoT systems from scratch composed of a toolbox with common building blocks of the IoT and a business process-based approach to orchestrate the setup and operation of these building blocks. This work includes a formalization of common IoT components into Building Blocks. Moreover, we developed a web application as a prototype, and we discussed our approach in means of our method applied into a Smart Factory case study.

1.2 Text Organization

The remainder of this work is organized as follows: Chapter 2 presents the fundamentals of this work and the related works. Chapter 3 presents the approach developed dividing in: toolbox and holistic method. Chapter 4 presents the web-based prototype, revealing the adopted architecture and technologies and describing the Front-End and Back-End implementation. Chapter 5 presents the evaluation of our approach, applying the proposed method and prototype to a real-world case study, also discussing its strengths and weaknesses. Chapter 6 concludes the work and summarizes our contributions.

2 FUNDAMENTALS AND RELATED WORK

In this chapter, we present the fundamental theoretical background referenced along with this study. First, we introduce IoT concepts, discuss the growing presence of IoT, the common architectures, and challenges emerging with this new paradigm. Second, we present the fundamentals concepts related to BPM. We also discuss the both BPM and IoT areas, their mutual benefits, and challenges. Finally, we present the related works.

2.1 Internet of Things

IoT is a global network that interconnects physical and virtual entities, or "things". Things are the basic building blocks and the main ingredient of the IoT (KHALED et al., 2018). IEEE Standard (2020) defines a "thing" as an IoT component or IoT system that has functions, properties, and ways of information exchange.

An entity is a particular thing, such as a person, place, process, object, concept, association, or event (IEEE, 2020). According to Bauer et al. (2013), a physical entity is a discrete, identifiable part of the physical environment that is of interest to the user for the completion of her goal. Physical entities can be almost any physical object or environment; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to closed or open environments. On the other hand, the same authors define a virtual entity as a computational or data element representing a physical entity.

The IoT pointed to the presence of smart environments, which embrace one or more physical entities sensing, acting, and automatically performing different tasks to enable their self-organization (HIRMER et al., 2016). Smart environments are divided into physical and digital environments. The physical environment contains devices, sensors, and actuators. The digital representation, also known as the digital twin (BOSCHERT; ROSEN, 2016), represents properties of the physical world based on a simplification of the world.

The development of the IoT leads to a wide range of applications in different domains where intelligent systems that obtain information from the physical world process such information and may perform actions on the physical world. Figure 2.1 illustrates the presence of IoT in people's life. In the following, some of the well-known smart environments are detailed.

Figure 2.1: Internet of Things different domains

Source: UFRN (2021)

Smart Homes (HARPER, 2006) can be defined as a residence equipped with computing and information technology which anticipates and responds to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond. According to Gomez et .al (2019), the foundation of such systems are home automation mechanisms, which provide the ability to monitor and control the building blocks of a home, e.g., windows, doors, electrical system, air conditioning system, energy production subsystem, alarm, appliances and so forth.

Smart Cities (CHOURABI et al., 2012) are built over different initiatives, including: management and organization, technology, governance, policy context, people and communities, economy, built infrastructure, and natural environment. According to Zanella et al. (2014), Smart Cities are about making better use of the public resources, increasing the quality of the services offered to the citizens, and reducing the operational costs of the public administrations.

Smart Factories (SHROUF; ORDIERES; G.MIRAGLIOTTA, 2014), also refered as Smart Manufactoring, Intelligent Factory, and Industry 4.0, refers to the shared vision of enhanced intelligence, flexibility, and dynamics over entire manufacturing processes and production. According to Bulik (2017), the Smart, and mostly digital, factory concept focuses on an integrated planning and monitoring process that includes product design, process planning, and overall integration and implementation of the manufacturing operation, making the manufacturing process more efficient and responsive to change.

There are several existing terminologies in the IoT area, but despite this variety, sensing and actuating objects are the most important for IoT. A sensor collects specific data within physical environments, and once a property has been observed and converted into a digital representation, the data/information is processed to create useful knowledge. On the other hand, an actuator device is used to perform actions upon the IoT environments according to the collected data converting information into action on a physical entity in the physical world, changing the state of the physical entity (IEEE, 2020).

In order to have a full understanding of this work is crucial to understand the difference between an IoT environment and an IoT system. An IoT system is "a system of entities (including devices, information resources, and people) that exchange information and interact with the physical world by sensing, processing information, and actuating" (IEEE, 2020). Differently, the IoT environment can be considered as the combination of IoT systems, networks connecting the IoT components, and any services that provide discovery, composition, and orchestration mechanisms.

2.2 Business Process Management

BPM is the discipline that aims to holistically operate, control, design, document and improve cooperative processes (DUMAS et al., 2018). Business process is a collection of events, activities, and decision points executed within application systems that are part of the real world involving actors and resources (humans, cooperative computer systems, and physical objects). Therefore, understanding the standard terms of a business process is crucial for its representation and interpretation, and these terms are listed and described below.

Activities can be seen as work that is done by participants of a process. As a type of activities, there are tasks and subprocesses. Tasks are atomic activities that have a time duration and represent units of work. They are performed by process participants, generally being human actors involved in the business process (DUMAS et al., 2018). Analogously, subprocesses are non-atomic activities.

Events correspond to elements that represent an atomic occurrence in the process, meaning they have no duration. They are generally used to specify the occurrence of a specific trigger that may throw a result or lead to the execution of a sequential step (DUMAS et al., 2018).

Decision points are points in time when a decision is made such that the execution flow of the business process is affected. There are different types of decision points: Exclusive decisions model the relation between two or more alternative activities, meaning that only one resulting alternative must be executed. Parallel executions occur when two or more activities do not have any order dependencies to each other, meaning that one does not require or exclude the other, thus, being able to be executed concurrently; Inclusive decisions model situations when the result of a decision may lead to one or more outcomes being executed. Rework and Repetition to repeat one or several activities, for instance, because of a failed check (DUMAS et al., 2018).

In order to model the processes, a commonly used language is the BPMN (OMG, 2014). The following subsection presents some of the most important features provided by BPMN.

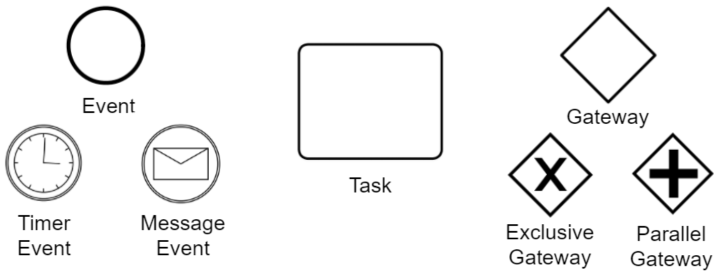
2.2.1 Business Process Model and Notation

Like any other language, a modeling language consists of four aspects: vocabulary, syntax, semantics, and notation (DUMAS et al., 2018). BPMN is a language with more than a hundred elements used to model business processes creating a bridge between process design and implementation. BPMN 2.0 is a standard for business process modeling that provides graphical notation easily understandable by users involved in the management, implementation, creation, and monitoring of business processes (OMG, 2014). The essential subset of BPMN objects categories are *flow objects*, *data*, *connecting objects*, *swimlanes*, and *artifacts*.

Flow objects are the main graphical elements to define the behavior of a business process (OMG, 2014). The *event* is something that "happens" during the course of a process. These *events* affect the flow of the model and usually have a cause (trigger) or an impact (result). An *activity* is a generic term for work that company performs in a process, and it can be atomic (task) or non-atomic (subprocess). A *gateway* is used to control the divergence and convergence of *sequence flow* in a process. Hence, it will determine branching, forking, merging, and joining of paths. The graphical representation of these objects are shown in Figure 2.2.

There are different types of *tasks* identified within BPMN to separate the types of

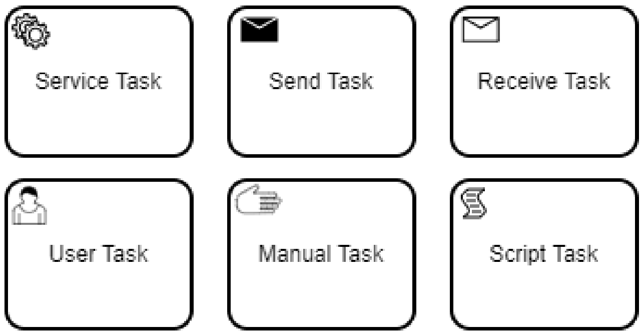
Figure 2.2: BPMN 2.0 Flow Objects



Source: The Authors

inherent behavior that *tasks* might represent. The list of *task* types may be extended along with any corresponding indicators. A Task which is not further specified is called *abstract task*. The mentioned *task* types are illustrated in Figure 2.3.

Figure 2.3: BPMN 2.0 Task Types

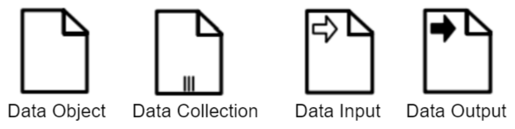


Source: The Authors

A *service task* is a *task* that uses some sort of service, which could be a Web service or an automated application. A *send task* is designed to send a *message* to an external *participant* relative to the process, and once the *message* has been sent, the *task* is completed. Analogously, a *receive task* is designed to wait for a *message* to arrive from an external *participant*. An *user task* is performed by a human with the assistance of a software application and is scheduled through a task list manager of some sort. A *manual task* is a *task* that is expected to be performed without the aid of any business process execution engine or any application, e.g., IoT expert positioning and installing a sensor at a specific location. Finally, a *script task* is execute by a business process engine with a defined script with a supported language.

Data objects provide information about what *activities* are required to be performed and/or what they produce. *Data objects* can represent a singular object or a collection of objects. *Data input* and *data output* provide the same information for *processes*. Figure 2.4 illustrates these objects.

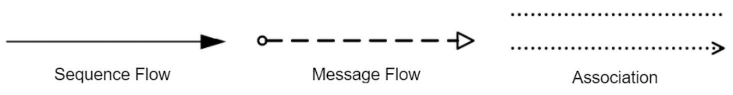
Figure 2.4: BPMN 2.0 Data Objects



Source: The Authors

The *connecting objects*, shown in Figure 2.5, enable *flow objects* to connect to each other or other information. A *sequence flow* is used to show the order in that *activities* will be performed in a *process*. A *message flow* is used to show the flow of *messages* between two *participants* prepared to send and receive them. An *association* is used to link information and *artifacts* with BPMN graphical elements where an arrowhead on the *association* indicates a flow direction.

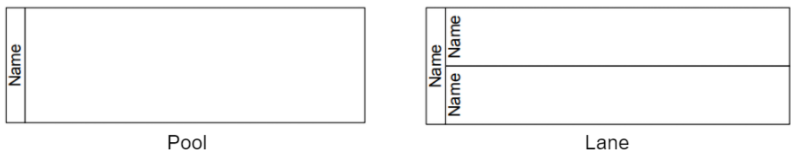
Figure 2.5: BPMN 2.0 Connection Objects



Source: The Authors

Within the *swimlanes* category, *pools* and *lanes* are meant to group the primary modeling elements. The *pool* is the graphical representation of a *participant* in a *collaboration*, and its shown on the left of Figure 2.6. It also acts as a *swimlane* and a graphical container for partitioning a set of *activities* from other *pools*. A *pool* may have internal details, in the form of the *process* that will be executed, or a *pool* may have no internal details, i.e., it can be a "black box". A *lane* is a sub-partition within a *process*, sometimes within a *pool*, and will extend the entire length of the *process* used to categorize and organize *activities*. The *lane* is depicted on the right of Figure 2.6.

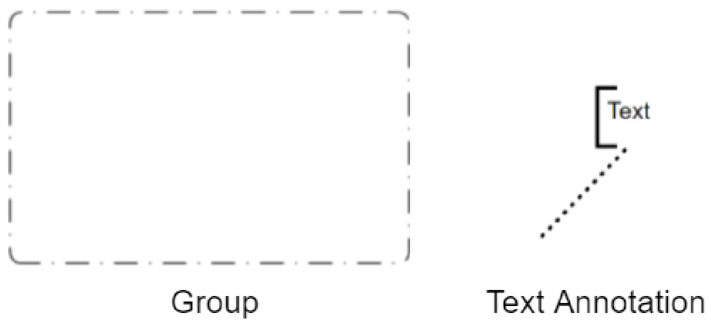
Figure 2.6: BPMN 2.0 Swimlanes



Source: The Authors

The *artifacts* are used to provide additional information about the *process*. A *group* is a grouping of graphical elements within the same category (this type of grouping does not affect the *sequence flows* within the *group*). *Text annotations* are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram. These elements are depicted in Figure Figure 2.7.

Figure 2.7: BPMN 2.0 Artifacts



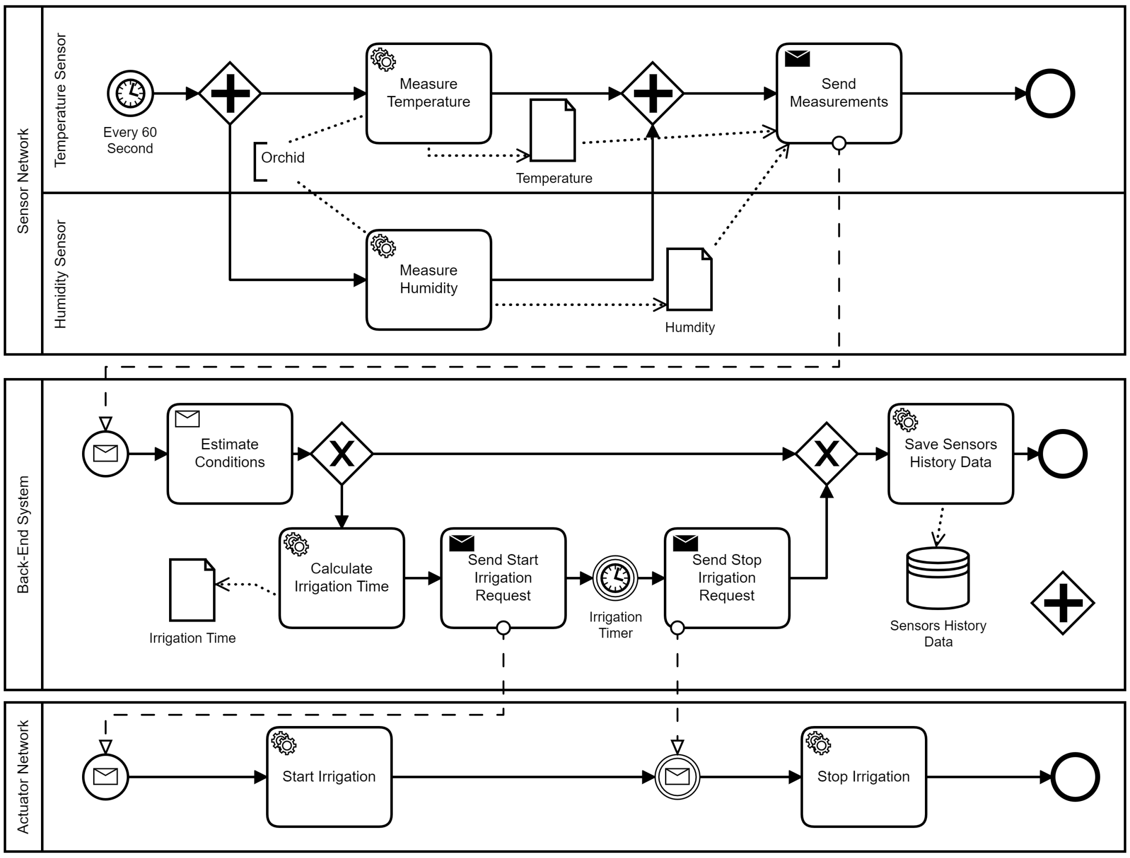
Source: The Authors

Finally, we present a process model with the described BPMN elements in Figure 2.8. The process illustrates a simplified, automated irrigation system use case. The system autonomously decides when to irrigate an Orchid based on the temperature and humidity measurements. It consists of three *pools*: the Sensor Network, the Back-End System, and the Actuator Network. The Sensor Network process is triggered every minute. After that, the parallel gateway splits the measurement of each variable to the respective sensor on its respective lane. Then, the temperature and humidity values are sent to the Back-End System *pool*. The message triggers the Back-End System process, and the first task evaluates if the irrigation is needed based on the sensed variables. For this, the exclusive gateway forces the process to follow only one of its paths. Suppose the temperature and humidity are below a defined threshold. In that case, it computes the irrigation time based on sensed variable levels, sends the request to the Actuator Network to start the irrigation process. The intermediate timer event waits based on the previous calculation and then sends a request to the irrigation to stop. Finally, the last task saves the sensors' data to the database.

2.2.2 Business Process Management System

Business Process Management Systems (BPMS) are designed to support processes at an operational level. It separates process logic from application code, creating an additional architectural layer and generally provides generic services necessary for modeling, execution, and monitoring processes.

Figure 2.8: Examplary of a simplified smart irrigation system



Source: The Authors

The BPMS can instantiate new process instances and control their execution based on the process model besides monitoring the whole progress of a process instance. Thus, business processes within BPMS are designed in a top-down manner as the processing logic is explicitly described in terms of a process model providing the schema for process execution.

In many scenarios, BPM approaches are used to automate processes through the support of contemporary BPMS, which uses connectors to established web communication protocols, e.g., HTTP. This way, with a suitable connector to an IoT protocol (SCHöNIG et al., 2020), it enables a communication architecture between process management and IoT devices, thus, some activities may be executed with the teamwork between software/hardware modules and humans.

2.3 Intersection of IoT and BPM

IoT devices are heterogeneous by nature and can aggregate and filter data, besides making decisions locally by executing parts of the business logic whenever central control is not required. Correspondingly, business modelers define processes using high-level languages, e.g., BPMN 2.0, needing to know just the domain but not specific knowledge to the IoT devices. Therefore, this decentralization requires design as well as execution time support.

Meaningful decisions in business processes require relevant information, and IoT devices are a consistent data source in this context. IoT devices can provide relevant data, such as events, in-memory databases, or complex event processing (CEP). Moreover, IoT devices can lead to more accurate data, reduced error, and efficiency gains since they could reduce the need to manually proclaim the completion of manual tasks with the availability of sensor data.

Considering a complex system scenario, such as a Smart Factory with self-driving vehicles, autonomous robots, and people moving around with localization tags as illustrated in Figure 5.1, the components interacting within this smart environment must be aware of the other components' locations and movements and interactions. Finally, such an environment might be better manageable linking the digital process with the physical world as enabled by the integration of IoT and BPM. For example, the completion of manual activities can be made observable through the usage of appropriate sensors. This way, IoT can complete BPM with continuous data sensing and physical actuation for improved decision making.

2.4 Related Work

In order to deal with the lack of standardization in the IoT area for the past years, several authors have contributed to the field with different approaches, architectures, models, ontologies, and frameworks. Moreover, many works are emerging in the literature combining the well-established BPM field and IoT. The related works were divided into a category focused on IoT contributions, including IoT architectures, models, and approaches to configuring environments. A second category discusses the works that combine both BPM and IoT fields. In order to summarize the focus of the analyzed works with the approach presented in this study, Table 2.1 shows each approach, its focus, and a brief description of the related work.

Table 2.1: Related work according to their focus

Authors	Focus	Approach
Our work	Configuration and Model	Provides a formalization of IoT components to building blocks and a process-based approach for configuration
(FRANCO DA SILVA et al., 2017)	Configuration	Setup of IoT environments using OASIS standard TOSCA
(MAYER et al., 2014)	Configuration	Goal-driven configuration of IoT environments for end users
(HIRMER et al., 2016)	Configuration and Architecture	Presents a method and a system architecture for automated provisioning and configuration of devices
(YELAMARTHI; AMAN; ABDELGAWAD, 2017)	Architecture	Presents an application-driven modular architecture
(JUSAS, 2017)	Model	Presents a feature-model based method for development of IoT-oriented applications
(OGC, 2008)	Model	Provides a semantically-tied means of defining processes and processing components associated of observations
(KHALED et al., 2018)	Model and Architecture	Provides a machine- and human-readable descriptive language for IoT devices and lightweight architecture
(BERMUDEZ-EDO et al., 2017)	Model	Provides a lightweight semantic model for IoT
(SURI et al., 2017)	Configuration	Presents a semantic framework for developing IoT-aware business processes

Source: The Authors

2.4.1 IoT: Architectures, models, frameworks and configuration approaches

Franco da Silva et al. (2017) present Internet of Things Out-of-the-Box, an approach using the OASIS standard TOSCA (OASIS, 2013) to automatically set up IoT environments. The goals are similar to ours: setting up IoT environments with as little effort as possible. Instead of a toolbox, they use a TOSCA Type Repository. However, the steps of defining requirements and selecting the most suitable TOSCA types are not described. Hence, it already needs to be known which hardware and software components are required to set up the IoT environment. Furthermore, the processes they use for setting up the IoT environments do not support human tasks. Thus, hardware device setup and other manual steps should already be done in their approach.

Mayer et al. (2014) present an approach that enables end-users to configure IoT environments at different places (e.g., workplace, home, and in public places), combining Semantic Web metadata and configuration of IoT devices. The configuration of the smart environment can be facilitated using a developed intuitive graphical editor that enables creating a model of the desired state of the user's environment. Furthermore, Mayer et al. lead with the complexity problem of configuring IoT environments by constraining end users' scope to the specific functionality offered by the graphical editor. Nevertheless, their approach does not use a process-based approach for the setup.

Hirmer et al. (2016) also have a goal similar to ours: facilitating the configuration of IoT environments even for domain users that are not familiar with all technical details. By providing basic information about the physical environment, their approach enables easy provisioning and configuration of devices. Furthermore, users' monitoring and managing of the IoT environments are enabled by its Digital Twin, i.e., the digital representation of the physical environment. However, the Hirmer et al. does not consider requirements to propose suitable software and hardware components neither a process-based setup.

Yelamarthi et al. (2017) introduce a modular IoT architecture. Their work is similar to our approach since they use modular building blocks to create an IoT architecture, addressing the challenges in selection of computational devices, wireless connectivity, internet gateway, and application cloud server to facilitate implementation in diverse applications. It shows that IoT architectures work best when built in a modular manner since replacing parts of the architecture becomes easier. We built on the idea of such a modular architecture and extended it by providing a means to set it up through a holistic lifecycle method based on our toolbox containing a large variety of building blocks.

In the past, many IoT environment models have been developed. These models contain information about the devices of the IoT environment, their properties (e.g., location or computing resources), the attached sensors and actuators, and their interconnection (FRANCO DA SILVA; HIRMER, 2020). However, these models highly differ in their content, the formats being used, and the domain they are applied to. Famous examples for such models are SensorML (OGC, 2008), IoT-DDL (KHALED et al., 2018), or IoT-Lite (BERMUDEZ-EDO et al., 2017). Large organizations maintain some of these models, and others have been created in research projects. Furthermore, some of them are even standardized. These models can be used to provide a standardized mean to describe the building blocks we aim for. Since many models are built based on ontologies (e.g., using the Web Ontology Language OWL (ANTONIOU; HARMELEN, 2004)), important aspects, such as hierarchies and inheritance, dependencies, and attributes, are already provided. Therefore, these models can be considered as a foundation for the building block design.

Jusas (2017) doctoral dissertation uses a feature modeling-based method for the development of an IoT-oriented application product line. The modeling of the IoT devices in many standard features is raised, and a Pareto optimal configuration of all possible IoT configurations satisfying the user requirements is elaborated. As we used our building blocks to deal with the IoT heterogeneity, this work used the development methods based on feature models to deal with the complexity of requirements and the heterogeneity of technology and environmental factors. Although this approach considers finding the most suitable environment according to specific requirements, no process-based setup is given.

2.4.2 IoT and BPM

In the literature, many works combining BPM and IoT are emerging. As shown by Janiesch et al. (JANIESCH et al., 2017), IoT poses challenges that will require enhancements and extensions of the current state-of-the-art in the BPM field. However, this integration can enable a better basis for such complex systems' planning, execution, and safety.

The work of Torres et al. (TORRES et al., 2020) introduces a systematic mapping study to find out how current solutions are modeling IoT into business processes. In their work, they refer to the well-known term IoT-aware business process as IoT-enhanced business process. They understand that business processes are more than informed or alerted by existing IoT elements but strengthened by its use, increasing their value and quality as a result. For example, business processes embracing IoT devices will be able to take real-world data into account to make more informed decisions, automate business process tasks, and improve their execution.

Suri et al. (SURI et al., 2017) present the IoT-BPO: a semantic framework for developing IoT-aware business processes. First, they formalize IoT resource descriptions in the context of business processes. Second, they formalize IoT properties and allocation rules for optimal resource management. Finally, the resource conflicts are resolved based on a set of pre-defined strategies.

2.5 Chapter Summary

In this chapter, we presented the main background of this study, first describing the main concepts and challenges of the IoT paradigm, emphasizing the present heterogeneity presented in the devices, protocols, and other related technologies. Following, we presented the discipline of BPM, and its fundamental concepts. Moreover, we presented some relevant BPMN 2.0 objects and discussed a exemplary of process model of a simplified smart irrigation to provide a practical example of the presented elements in this section. Then, notwithstanding, we presented the BPMS concept. Also, we discuss the mutual benefits of IoT and BPM and which challenges are tackled with the intersection of both areas.

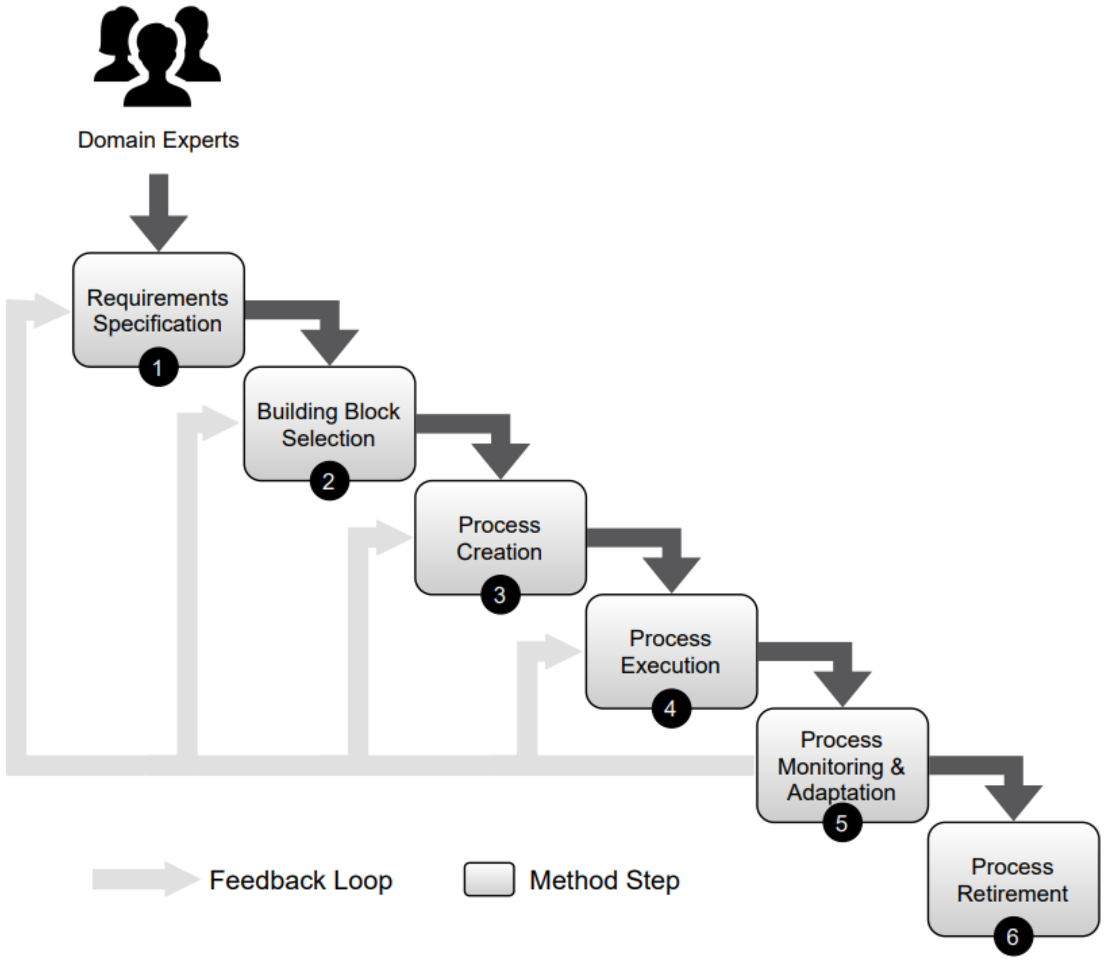
Moreover, in this chapter, we presented numerous related works divided into two different categories. The first category focuses on the IoT field contributions and contains different architectures, models, and methods related to our approach. Approaches such as Hirmer et al. (2016) and Mayer et al. (2014) focus on facilitating the configuration of IoT environments in some aspect. The first one enables easy provisioning and configuration of devices by providing basic information about the physical environment and monitoring the implemented system through a digital twin. The second one offers a graphical editor constraining the users' scope but reducing the complexity. Franco da Silva (2017) goal is to set up IoT environments with as little effort as possible using the OASIS standard TOSCA to automatically configuration. The approach from Yelamarthi et al. (2017) introduces a modular IoT architecture to tackle the complexity implementing diverse applications. The BBs and BBIs of our Toolbox were built on the idea of such a modular architecture and extended through a holistic method. In addition, Jusas (2017) a feature modeling-based method to model IoT devices in many standard features, similiar to what we did to model IoT devices to our BBs and BBIs. Furthermore, some models are presented, such as SensorML (OGC, 2008), IoT-DDL (KHALED et al., 2018), or IoT-Lite (BERMUDEZ-EDO et al., 2017), which were used to inspire the modeling of our building blocks.

3 HOLISTIC METHOD FOR CONFIGURING IOT ENVIRONMENTS

This chapter presents a holistic method that describes the necessary steps domain experts need to undertake to set up IoT environments and their applications from scratch. The method is depicted in Figure 3.1 and detailed in Section 3.2.

The foundation for this method is our toolbox, containing a predefined set of building blocks that can be combined to set up the desired IoT environment and a predefined set of requirements to help domain experts select suitable building blocks.

Figure 3.1: Method for setting up IoT environments based on our toolbox



Source: The Authors

Even though our method is generic in terms of the kinds of applications that should be set up, applying it specifically to the IoT domain emphasizes its strengths. In the IoT, many heterogeneous software and hardware components usually need to be set up, overwhelming application developers. Our method aims to reduce the effort for setting up IoT environments and applications by decreasing the necessary domain knowledge, since the technical details are abstracted through the building blocks.

The remainder of this chapter is structured as follows: In Section 3.1, we introduce the toolbox and its definitions. Section 3.2 details the method describing step-by-step. Finally, in Section 3.3, we summarize the current chapter.

3.1 Toolbox

The toolbox serves as the foundation for our method. It offers a set of building blocks (BB) that are divided into different categories. These categories include, for example, sensor, actor, communication, gateway, or platform. The BBs are provided by experts in building IoT applications and are provided by the toolbox, which domain experts can then access.

In order to ensure understanding and applicability of our approach, we introduce a formalization of the toolbox. A BB is formalized as follows:

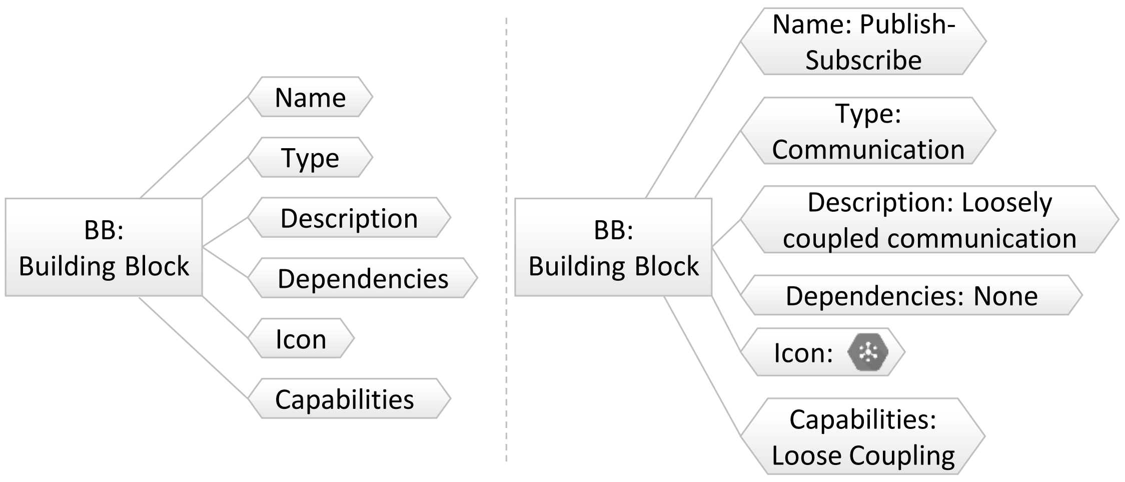
Definition 3.1.1 (Building Block)**.** Let $bb \in BB$ be a tuple $bb := (Name, Type, Icon, Description, Dependencies, Capabilities)$, whereas BB is the set of all available building blocks in the toolbox TB. A bb has the following properties:

- $Name \neq \emptyset$: unique name of the building block.
- $Type \neq \emptyset$: the type $t \in T$, where T is the set of available bb types.
- $Icon \neq \emptyset$: icon of the building block.
- $Description$: optional description.
- $Dependencies$: set of $bb \in BB$ related to this building block.
- $Capabilities$: set of capabilities of the building block, used to map user requirements to the BBs of the toolbox.

Figure 3.2 depicts the structure of a building block on the left. Each block contains a name, a type, a description (suitable for domain experts), dependencies to other building blocks, an icon to ensure recognition, and a set of capabilities that can fulfill the users' requirements. The example on the right of Figure 3.2 is a BB for the communication paradigm publish-subscribe, enabling loosely coupled communication.

In an initial phase, experts in the IoT domain need to agree upon a common list of building blocks, which must be extensible, since new technologies frequently appear in the IoT. We are aware that this is an ambitious goal, which would require some sort of standardization IoT experts agree upon. In our vision, IoT technology providers create

Figure 3.2: Left: Remainder of a Building Block, right: Building Block example for publish-subscribe communication



Source: The Authors

building blocks themselves in order to promote their new solutions and add them to the toolbox. With a strong community, the toolbox could grow to a comprehensive collection of all kinds of IoT components.

Each BB has one or more implementations, referred to as BBI in the following. A BBI is a concrete implementation of a BB, which is formalized as follows:

Definition 3.1.2 (Building Block Implementation)**.** Let $bbi \in BBI$ be a tuple $bbi := (Name, Artifact, Interface, Description, Icon, ImplementedBB, Dependencies)$, whereas BBI is the set of all available building block implementations in the toolbox TB. A bbi contains:

- $Name \neq \emptyset$: unique name of the building block implementation.
- $Artifact$: software artifacts.
- $Interface$: interface description.
- $Description$: optional description.
- $Icon \neq \emptyset$: icon of the building block implementation.
- $ImplementedBB \neq \emptyset$: a list of BB implemented by the building block implementation.
- $Dependencies$: list of $bbi \in BBI$ related to this building block implementation.
- $Features$: set of features of the BBI, used to map user requirements to the BBIs of the toolbox.

Furthermore, $bbiMapping : BB \to BBI$ corresponds to the mapping, which assigns a BB to one or more BBIs.

A BBI can contain a software artifact, which can be of different types, e.g., a Docker container [1], a cloud service [2], or a binary application. Furthermore, BBIs can have a set of dependencies that could require the installation of different BBIs. For example, some BBIs might require installing certain programming languages (e.g., Java, Python) or a specific operating system. In addition, each BBI requires a definition of its interfaces to enable easy orchestration via business processes. In the case of software artifacts, these interface descriptions should be defined based on standards, such as WSDL (Web Services Description Language) (WEERAWARANA et al., 2005).

This set of attributes should describe the building block implementations and how they can be managed and configured. Also, the different communication languages they support, and eventually, the relevant IoT semantics of how a BB or BBI can use another building block.

For our BB example in Figure 3.2, implementations include MQTT brokers, such as Mosquitto (LIGHT, 2017) or RabbitMQ (RABBITMQ, 2007). Once domain experts decide to use the publish-subscribe paradigm, they will find the according building block in the toolbox and will be able to select one of the corresponding implementations.

Moreover, a significant feature to aid the user experience using the toolbox is the recommendation of the BBs based on a set of requirements elected by the user. This recommendation should be able to map stakeholder's requirements to BB's capabilities and/or BBI's features. The toolbox contains a predefined set of requirements which can either be a functional requirement (e.g., *Sensing Accuracy*) or a non-functional requirement (e.g., *Security*).

In case the requirement is measurable, it must have a parameter attached to it. For example, the user can select the requirement *Data Transfer Rate* and some parameters might be required as input (e.g., at least 300 kbits/s). However, if the user selects a non-functional requirement, there is usually no need for parameterization. Therefore, the requirements and the requirements parameters are formalized as follows:

Definition 3.1.3 (Requirement)**.** Let $req \in Req$ be a tuple $req := (Name, Type, Parameter)$, whereas Req is the set of all available requirements in the toolbox TB. A req has the following properties:

- $Name \neq \emptyset$: unique name of the requirement.
- $Type \neq \emptyset$: the type $t \in T$, where T is the set of available requirement types.

[1]https://www.docker.com/resources/what-container
[2]https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services

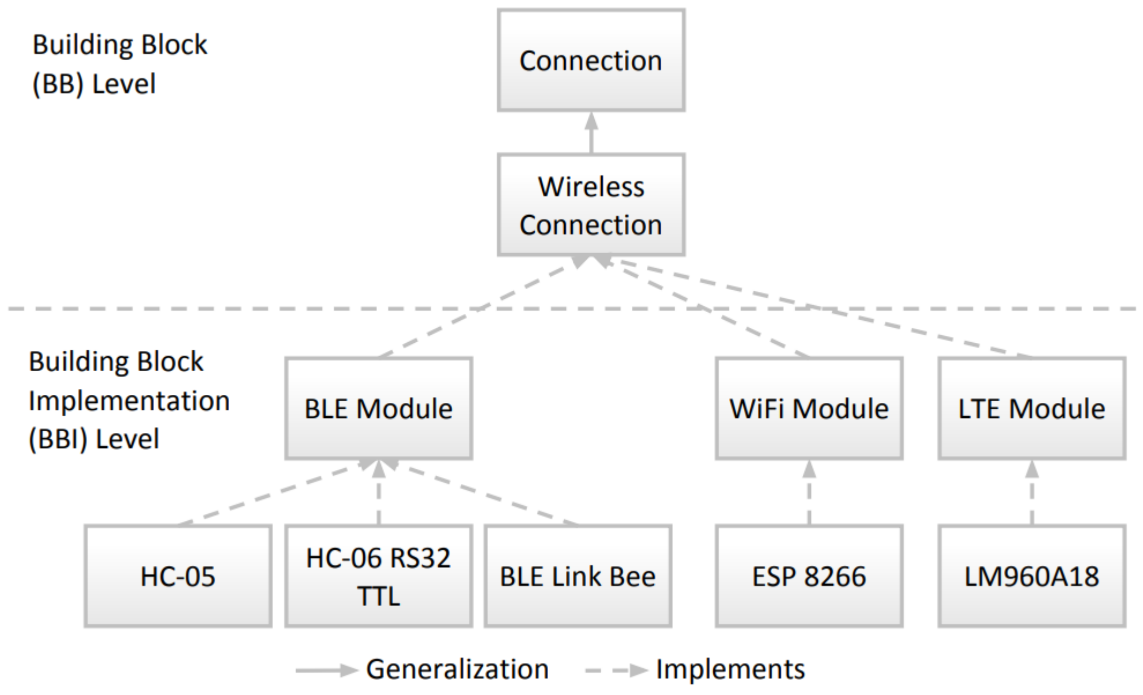
- $Parameter$: optional parameter.

Definition 3.1.4 (Requirement Parameter)**.** Let $parameter \in req$ be a tuple $parameter := (Label, Unit, Operator, Value)$, whereas req is $req \in Req$ in the toolbox TB. A $parameter$ has the following properties:

- $Label \neq \emptyset$: name of the requirement parameter.
- $Unit \neq \emptyset$: unit of the requirement parameter.
- $Operator$: optional operator $op \in Operators$, where $Operators$ is the set of available operators.
- $Value$: optional value of requirement parameter.

Finally, the BBs and BBIs can be defined in a hierarchical structure, meaning that building blocks can inherit characteristics of other building blocks, enabling their specialization. The depth of the hierarchy is arbitrary and is decided upon by the designer of the toolbox. An example of the Wireless Connection BB hierarchy is depicted in Figure 3.3.

Figure 3.3: Example of hierarchy of Wireless Connection BB



Source: The Authors

3.2 Holistic Method for configuring IoT environments

After the toolbox is filled with BBs and BBIs, it can serve as a foundation for our method to set up IoT environments. The following paragraph summarizes all the method steps.

In step 1 of this method, domain experts and involved stakeholders discuss their IoT environment or application requirements and define a set of requirements for the application. The set of requirements is defined as $R \neq \emptyset$, whereas each $r \in R$ describes a specific application requirement. After that, in step 2, the toolbox recommends building blocks selected by domain experts. In step 3, a business process is created by experts, using the BPMN 2.0 notation, based on the selected building blocks, defining the necessary steps that need to be undertaken to set them up. In step 4, this process is executed through a BPMS, guiding the domain experts in creating the IoT environment for their application. In step 5, the IoT environment is already set up, and the application is running. This step allows a continuous search for changes to the IoT environment for error detection or for considering new requirements. In the final step 6, the IoT environment is retired once an IoT application reaches its lifetime. In the following, the toolbox and the steps of this method are described in more detail.

Additionally, each step has one or more feedback loops to the previous steps, ensuring that new requirements or changes in the selection of building blocks can be done throughout the whole method.

Step 1: Requirement Specification

In the first step of our method, IoT application developers, i.e., the domain experts and their stakeholders, need to define a set of requirements for their application.

As discussed in the previous section, a *requirement* is a condition or capability needed by the user to solve a problem or achieve an objective. These requirements in an IoT system's context can be functional (e.g., sensed variable, sending rate) or non-functional (e.g., lifetime, reliability), typically referred to as quality-of-service (QoS) (BALEN; ŽAGAR; MARTINOVIC, 2011) parameters in the IoT domain.

A service in IoT can be defined by the combinations of *"functionalities, interoperability, interactions, communication abilities, related data and ability to use the related data"* (BHADDURGATTE; B.P, 2016) of devices for implementing the IoT system to meet the requirements of specific applications and end-user systems.

Our toolbox provides a list of predefined requirements for the most common IoT application scenarios where the stakeholders can find the ones that fit their needs. The list should be kept up-to-date to the current state-of-the-art of the IoT field. An IoT application can be composed of various components (e.g., different devices, communication protocols) that have their particular specializations and can come to satisfy one or more requirements.

The heterogeneous nature of IoT hinders the specification of requirements. As a consequence of the multi-layered architecture of IoT systems, the requirements need to be specified differently for each layer (COSTA; PIRES; DELICATO, 2017). To cope with this, the requirements in the Toolbox are divided into types (e.g., sensing, acting, communication), and each type is mapped to one or more categories of BBs.

These requirements need to consider different aspects, such as network capabilities, used communication paradigms, costs, efficiency, security, privacy, available computing resources, and so on. Therefore, defining such requirements might involve many different stakeholders and even a requirements engineer, i.e., a person who would understand the domain of the desired application and possess enough IoT knowledge to be aware of the convenient features and resources for the development.

To enhance the reusability of the toolbox in multiple projects, a systematic collection of requirements and the respective chosen building blocks are documented to provide an overview of possible relations, assisting in requirements engineering for future applications.

Step 2: Building Block Selection

In the second step of our method, involved stakeholders select the building blocks they need based on the set of requirements created in step 1. The toolbox will contain a collection of best practices in IoT application development and, hence, give an overview of which technologies and approaches are available without the need for domain experts to acquire this knowledge themselves.

The selection of BBs is made based on recommendations by the toolbox, which are generated by matching the requirements of the IoT application to the capabilities of the building blocks and the features of the building block's implementation. The involved stakeholders conduct the final selection. We define the mapping that assigns a requirement

Table 3.1: Example of BB and BBI selection.

Building Block	**Matching Requirement**	**Implemented by BBI**
UWB localization	indoor localization	RTLS localization modules
BLE localization	indoor localization	BLE modules
WiFi	wireless communication	WiFi network
LTE	wireless communication	4G network
5G	wireless communication	5G network

Source: The Authors

to one or more building blocks as $reqMapping : R \rightarrow BB$.

To make this selection process clearer, let us assume that a company aims to develop a smart factory application in which self-driving vehicles transport materials on the shop floor. The goal of this application is that these vehicles dynamically determine the best routes and consider other vehicles and people, and utilities of the shop floor. To achieve this, the vehicles need to be equipped with sensors to monitoring the environment and need to communicate with other vehicles and participants on the shop floor through a wireless network. Setting up such a complex scenario requires the expertise of many different stakeholders that need to agree upon functional and non-functional requirements. Regarding functionality, for example, an indoor localization system is essential to monitor the position of vehicles, utilities, and people on the shop floor. In addition, a broad-band wireless network needs to be established to cope with the high data load. Furthermore, non-functional requirements are of high importance in such safety-critical environments. Security, safety, robustness, accuracy, and real-time capabilities are only some of the essential requirements of this scenario. In workshops, the stakeholders need to assess and discuss the requirements.

Based on the functional requirements “indoor localization" and “wireless communication", we show in Table 3.1, which BBs and BBIs could be recommended by the toolbox.

If we now consider the non-functional requirements, such as high efficiency, robustness, and accuracy, the toolbox would filter the resulting BB list accordingly. For example, localization using Bluetooth Low Energy (BLE) and triangulation has issues regarding accuracy and also efficiency. Hence, the BB “BLE localization” would not be recommended by the toolbox. Similar issues occur for WiFi connections since they tend to be unstable. Instead, the toolbox could suggest using Long Term Evolution (LTE) (COX, 2014) or 5G (TOWNSEND et al., 2014) networks. The company can then choose depending on costs or already available infrastructure on the shop floor.

This example illustrates how requirements and building blocks fit together. After the set of BBs and BBIs is retrieved from the toolbox, the domain experts and their involved stakeholders need to choose among the BBs. Once the BBs are selected, they need to choose from the BBIs the most fitting ones. This decision is usually made based on the (e.g., hardware) costs or available expertise.

Step 3: Process Creation

The third step deals with the business process modeling (DUMAS et al., 2018) to guide domain experts and involved stakeholders through the process of setting up their IoT environment and applications. The tasks of the process set up the previously selected BBIs, whereas usually several process tasks are required for each BBI. The business process creation needs to be conducted manually by experts, which can become a cumbersome and time-consuming task.

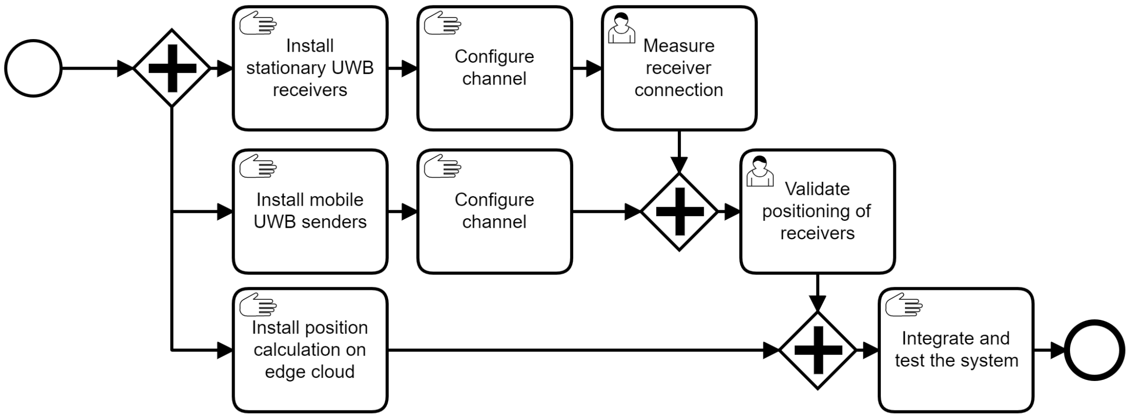
Even though our approach deals essentially with the configuration of the IoT environment, it gives the foundation for the creation of an IoT-aware process, helping to deal with some of the commonly tackled challenges when exploiting applications using BPM and IoT. For example, in order to collect all relevant data, sensors need to be carefully placed, whereas the placement of the device is part of the environment setup.

Hence, setting up a given BBI, in most cases, will require common steps to be taken even in different scenarios, and the toolbox categorizes these typical steps as *best practices* that can be used for the process creation. Assume, for example, that a BBI sets up an indoor localization system on the shop floor. The corresponding steps that need to be undertaken are mostly the same for different companies with only minor differences based on the layout of the shop floor.

An example of a process is provided in Figure 3.4 in BPMN 2.0 syntax. As discussed above, this process shows a simplified version of the different steps to set up an indoor localization system using UWB (ULLAH et al., 2009) technology. In parallel, the receiver and sender hardware need to be installed on the shop floor. Furthermore, software for the position calculation needs to be installed in an edge cloud environment, which will later communicate with the receivers. Finally, after configuration and measurements of the hardware, the system can be integrated, tested, and then go into production.

Note that the configuration of different BBIs can include multiple tasks in the process due to its complexity, as shown in our example process. For manual creation, the

Figure 3.4: Exemplary simplified BPMN 2.0 process to set up the indoor localization system.


Install stationary UWB receivers
Configure channel
Measure receiver connection
Install mobile UWB senders
Configure channel
Validate positioning of receivers
Install position calculation on edge cloud
Integrate and test the system

Source: The Authors

best practices can help in selecting the tasks for the creator. Which tasks can be conducted in parallel needs to be decided by the creator as well. Hence, some domain knowledge is required for process creation. In doubt, the steps should be conducted sequentially.

The toolbox allows for a huge variety of components to be selected, with different levels of complexity regarding its manipulation. For example, there are plug-and-play devices embedded with sensors and MCU (microcontroller unit) on the one hand and, on the other hand, the user can also select a sensor module for the application setup, that will need the assembled into a development board, like Arduino or users can build a totally custom board in case they have the engineering skills. It is not a concern of the Toolbox how the user will wire up the modules or build the custom board.

Step 4: Process Execution

In step 4, the process is then executed to realize the IoT environment's setup. For BPMN 2.0, for example, the established BPMS Camunda could be used. In each step of the process, domain experts get notifications about the tasks they need to conduct, for example, plugging in sensors, configuring a WiFi connection, or installing software on IoT devices. Documentation of the different technologies that are set up can be found in the BBIs. In simple scenarios, setting up the IoT environments should also be possible for non-expert users. However, usually, this requires domain-specific expertise, as shown in our Smart Factory example. After task completion, the process moves to the next task. Parallelism is usually supported by such BPMS as well.

The toolbox provides an interface allowing the user to visualize the big picture of the process and show the current tasks that need to be completed. Depending on the task type, the user can provide parameters when needed. Most of the current BPMS will be supported in future work, and the Back-End will encapsulates the processing and the logic of the chosen BPMS for the respective process. For example, if the process modeler chose Camunda (CAMUNDA, 2021), the Back-End will be prepared to communicate with the workflow engine through the Camunda Rest API, i.e., PUT, POST, and GET HTTP requests.

Once the process has successfully reached its end, the IoT environment is set up. This process can then be reused in similar scenarios, sometimes only with minor necessary adaptations. However, in case of issues in the process creation or execution, e.g., due to unforeseen errors, experts need to be available to cope with occurring difficulties and fix the problems. All occurred problems should then be documented inside the BBI's description so that this knowledge is preserved.

Step 5: Process Monitoring and Adaptation

In this step, the IoT environment is already set up, and it is monitored in case of some bad behavior needing some adaptation or a change in the requirements. It is not unusual that adaptations are necessary overtime after an IoT environment was set up. For instance, due to changes in the application and, hence, changes in the requirements or due to failing devices that need to be replaced.

Our toolbox does not provide a monitoring strategy for IoT environments and applications. The monitoring feature is part of future work and the goal is to integrate the prototype with an IoT platform like Azure [3] or MBP [4].

If adaptations are necessary, the feedback loops in our method allow us to return to one of the previous steps, redefining the application's requirements, adding or removing requirements, changing the selection of BBs and BBIs, or executing the process for configuring the environment again.

In case of changes in the selection of BBs and/or BBIs, the process creation step then creates a so-called *adaptation process*, which includes removing unnecessary BBIs and only setting up and integrating the newly added ones. After executing the *adaptation process*, the IoT environment is set up again, considering the new requirements.

[3]https://azure.microsoft.com/
[4]https://github.com/IPVS-AS/MBP

Step 6: Process Retirement

The final step is the retirement of the IoT environment, which is executed once an IoT application reaches its lifetime. In this case, creating a *termination process* is required, reversing the steps of the original process setting up the environment.

Using the process for creation as a foundation eases the conception of the termination process. Providing a termination process together with the creation process in Step 3 is a best practice using the Toolbox to ease the execution of the step. However, it is known that after some time running the IoT environment, some tasks might change in the provided termination process.

After this process is executed, the IoT environment is retired. Note that the creation and termination processes should be stored since they can be helpful to re-setup IoT applications after some time.

3.3 Chapter Summary

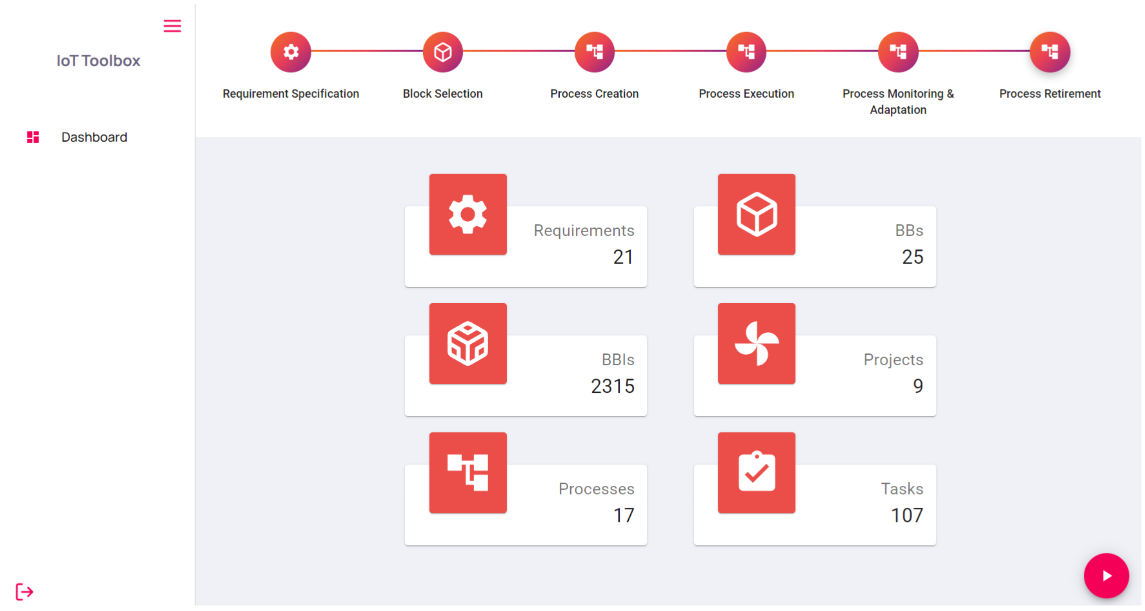
In this chapter, we presented our contributions. In Section 3.1, we describe our toolbox that serves as the foundation for our method. Along this section we introduce the formalization of the definitions of the toolbox: *Building Block*, *Building Block Implementation*, *Requirement*, and *Requirement Parameter*.

Subsequently, we present our holistic method and detail each step. In the *Requirement Specification* step, stakeholders define the requirements of the IoT environment. The toolbox provides a pre-defined set of requirements to be selected by the users. In the *Building Block Selection* step, the BBs and BBIs are selected based on the previously defined requirements. The recommendation of BBs and BBIs is made by the Matching Algorithm, which is based on the defined requirements. The *Process Creation* deals with the business process modeling to guide the stakeholders through the process of setting up their IoT environments and applications. Afterward, in the *Process Execution* step, the modeled process is executed on a BPMS process engine in order to realize the IoT environment's setup. Then, in the *Process Monitoring and Adaptation* step, the configured environment is monitored, and adaptations could be made through an *adaptation process* request. Finally, in the *Process Retirement* step the *termination process* is executed to retire the IoT environment.

4 PROTOTYPE

This chapter describes the development of a practical application of the approach presented in Chapter 3. The web-based prototype was developed to serve as a proof-of-concept for our toolbox and method. The prototype is available on Github [1]. The homepage of the prototype is depicted in Figure 4.1.

Figure 4.1: BBI selection example



Source: The Authors

In the first section of this chapter, the architecture of the prototype is described, designating the architectural pattern and the reasons of using it. The technologies section then introduces the technologies used for the development of the prototype, including the programming languages and the motivation of using for this context. In the following sections, the *server component* and the *client component* are respectively described, additionally presenting its UML class diagrams and sequence diagrams (SOMMERVILLE, 2015). Finally, the chapter summary section summarizes this chapter.

4.1 Architecture

The first step for developing the prototype was the definition of the architecture based on the pre-established requirements: modularity, reusability, and scalability. The modularity eases the individual development of the prototype's features, i.e., each method

[1] https://github.com/IoToolbox

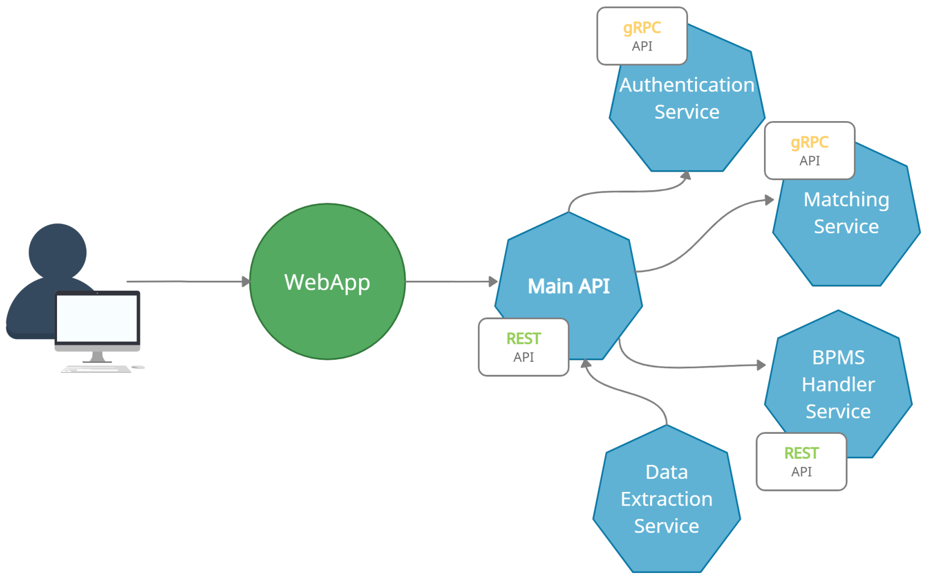
step and other complex features, such as the requirements matching. Reusability is a good software development practice and improves the software product development process. Finally, scalability is important because we aim to use the prototype as a foundation for a large-scale open-source IoT platform in future work.

Considering all the prototype requirements, the Microservice Architecture (MSA) (NEWMAN, 2015) was chosen. A Microservice Architecture is a distributed application where all its modules are microservices. First, a microservice realizes a distinct architectural capability and exhibits a high degree of independence regarding development and operation (RADEMACHER; SACHWEH; ZüNDORF, 2017). Thus, the Microservice Architecture denotes an architectural style that is focused on building distributed software systems as sets of specialized, autonomous services (NEWMAN, 2015), which communicate with each other through lightweight mechanisms (i.e., a RESTful API).

The Microservice Architecture does not favor or forbid any particular programming paradigm and the advantages are commonly accepted both in academia and industry, i.e., maintainability, reusability, scalability, availability and automated deployment (ALSHUQAYRAN; ALI; EVANS, 2016).

Applying the Microservice Architecture to our approach, we isolate the complex tasks of the project into different and independently developed microservices. Some complex services are the matching between the set of selected requirements and the building blocks, the BPMS interoperability, and other simple tasks, like user authentication in the platform. Each microservice is detailed in Section 4.3.

Figure 4.2: Proposed MSA architecture of the prototype



Source: The Authors

Figure 4.2 depicts the prototype architecture and its microservices, where the WebApp (Front-End) is the entry point for the user to the prototype. Then, all requests from the Front-End to the Back-End first go to the Main API through the REST API. Then, suppose the request is forwarded to another microservice. In that case, the request communicates with the desired module via the exposed API, e.g., communication with the Matching Service is done via gRPC API, while BPMS Handler Service is via REST API. However, the Data Extraction Service does not expose an API but only communicates with the REST API of the Main API module.

4.2 Technologies

The key motivation for developing the prototype is to ease, graphically and interactively, the entire process of configuring IoT environments and applications. Therefore, the prototype features aim to go through the presented method step-by-step, ranging from requirements elicitation and building blocks selection to the functional configuration of the environment with the support of BPM.

Considering the opportunity provided by the Microservice Architecture, we chose the technologies, programming languages, and libraries for each microservice and the application interface individually, taking into account the service and function implemented by the microservice and the developer's familiarity with these resources.

The main programming languages used in the prototype development are JavaScript and Python. JavaScript was initially created for developing client web applications running in the browser, but nowadays, it can also be run on the server-side. Therefore, working with JavaScript allows the developer to program for the client- and server-side using a single programming language, enabling the reuse of components and resources (MOZILLA, 2015). This, in line with the developer's familiarity with the language, was the motivation to implement most of the microservices and the graphical interface using the respective programming language, detailed in Section 4.3 and Section 4.4, respectively.

Python programming language was used to implement services that require more processing and are somehow more complex. Millman and Aivazis (2011) state that Python has become the standard language for exploratory, interactive and computation-driven research. Moreover, according to Oliphant (2007), the Python programming language stands out as a scientific computing platform for reasons such as an open-source license to use and distribute any Python-based applications; no concerns about portability as it works across platforms; clear syntax and sophisticated constructs that allow for object- or procedure-oriented code; ability to interact with a wide variety of other software components.

A key technology for the developing of the prototype is Docker (Docker, 2021). According to Jamillo et al. (JARAMILLO; NGUYEN; SMART, 2016), Docker has been a disruptive technology which changes the way applications are being developed and distributed. With a lot of advantages, this technology is a very good fit for implementing microservices architecture. Docker is an example of a container service that is based on operating system virtualization and Linux container services. It encapsulates the microservice in a Docker container, which can then be maintained and deployed independently. Each of these containers will be responsible for running a specific business function, i.e., an individual microservice (RAJ; JASMINE, 2021).

Microservices should be independent components, each having their own isolated logic being equipped with dedicated memory persistence tools. In a Microservice Architecture, the database is defined as a distributed-type database rather than a centralized database, meaning the individual choice of database technology for each microservice. The microservice's dedicated databases in our prototype use MongoDB (MongoDB Inc., 2021): an open-source document-oriented NoSQL database which allows high efficiency, scalability, and data replication.

Moreover, the microservice's communication with each other is conducted through lightweight mechanisms, using the Representational State Transfer (REST) mechanism, an architectural style that enables access to business logic in abstract resources at Universal Resource Identifiers (URI) using HTTP. REST facilitates loosely coupled system development by providing JavaScript Object Notation (JSON), a lightweight data-interchange format. Besides, the gRPC (Google, 2021) mechanism is another architectural style. It is a modern open-source, high-performance Remote Procedure Call (RPC) framework founded by Google. It uses the message format *protobuf*, which is highly compact and efficient to serialize structured data.

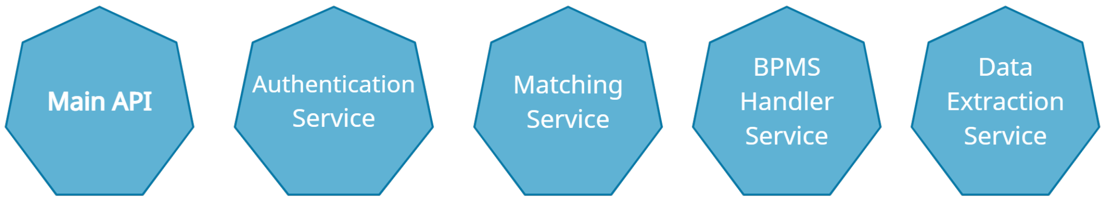
Both mechanisms enhance reusability and flexibility in the applications. However, the most important advantage of using these architectures is that a single application can seamlessly interact with the software developed in many other languages, maintaining the decoupling from the implementation platform.

In the following sections, Back-End (i.e., microservices, APIs, and databases) and the Front-End (i.e., the interface of the prototype) are detailed.

4.3 Back-End

The Back-End is responsible for the business logic implementation of the prototype and it is composed of the main API and a few microservices, besides the BPMS integration for process management and execution environments. All the Back-End modules are illustrated in Figure 4.3 and summarized in Table 4.1. Each of them is presented in the following subsections.

Figure 4.3: Back-End Microservices



Source: The Authors

Table 4.1: Back-End Microservices descriptions

Microservice	**Descriptions**
Main API	The gateway between client- and server-side. Also, CRUD operations of Toolbox models.
Authentication	Implements the prototype's authentication flow.
Matching	It is responsible for implementing the requirement, BBs, and BBIs matching algorithm.
BPMS Handler	The gateway between the prototypes backend to a BPMS. It processes and translates the prototypes logic to a target BPMS and communicates with it.
Data Extractor	Responsible for gathering, filtering, and processing data to BBs and BBIs models.

Source: The Authors

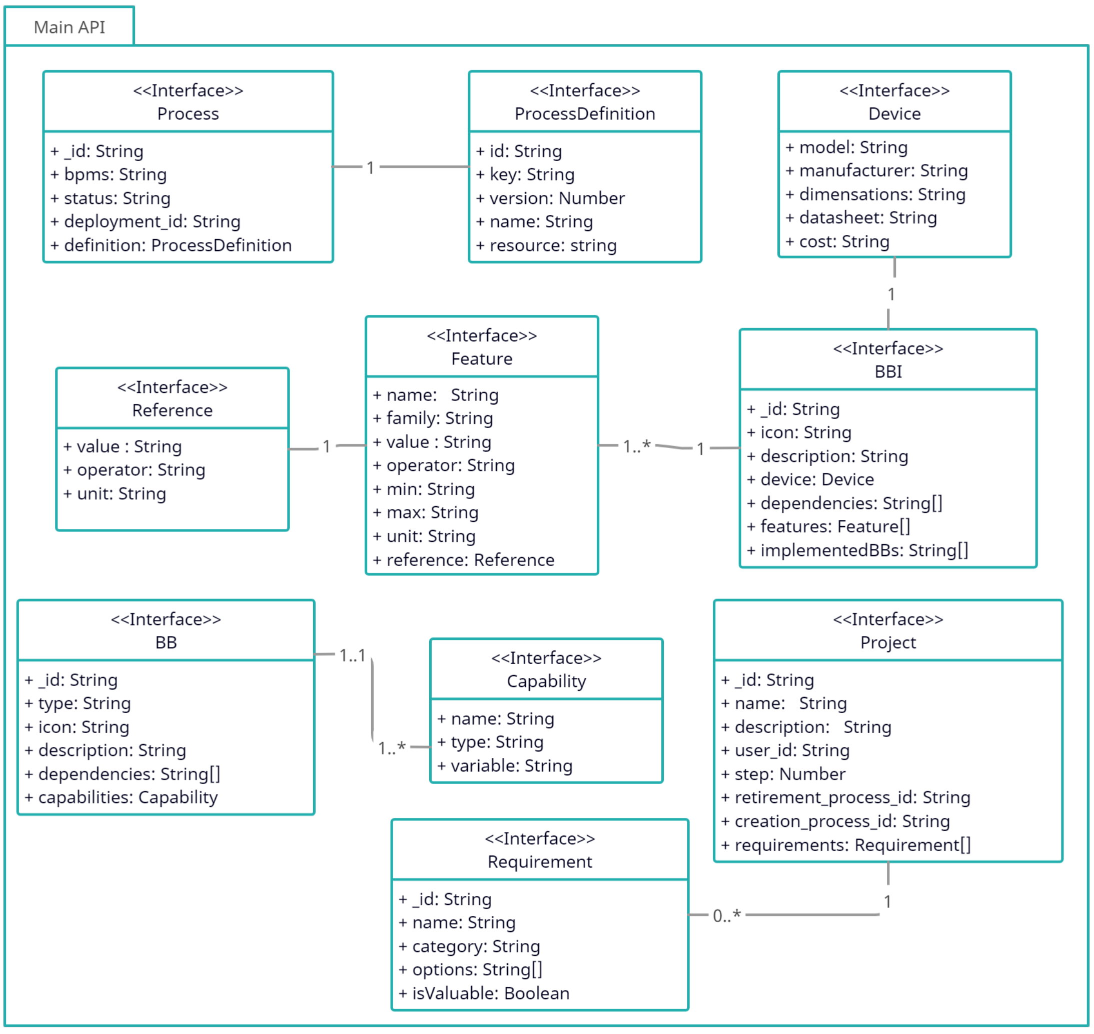
4.3.1 Main API

The main API is implemented in NodeJS (OpenJS Foundation, 2021), an open-source JavaScript server environment with a REST-based program interface. It follows the *Back-End for Front-End* Microservice Architecture pattern (PAVLENKO et al., 2020), meaning a microservice implemented that serves as a gateway for requests from the Front-

End.

As the entry point of the Back-End, this module serves as the orchestrator between the Front-End and the Back-End interacting via REST-API. The requests are processed or forwarded to some of the microservices (e.g., Matching Service), which means the execution of basic CRUD operations for the application models or performing requests to another application's microservices and combining responses in case of one client's action leads to this case.

Figure 4.4: Main API class diagram



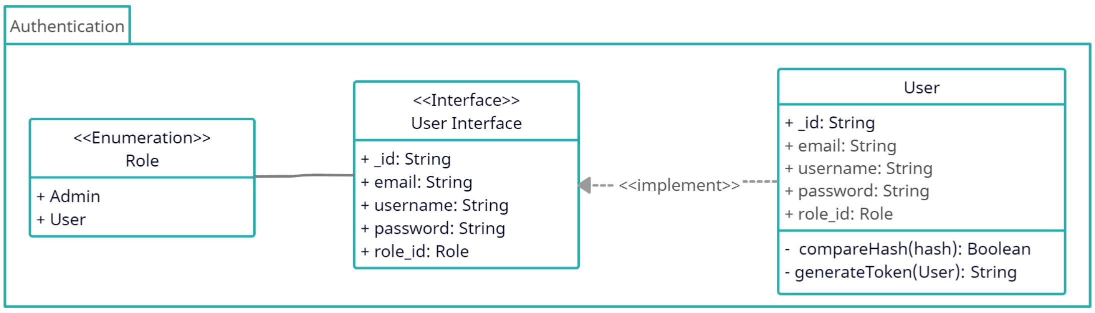
Source: The Authors

Figure 4.4 shows the class diagram of the Main API microservice. The main models of the prototype are managed by this component, meaning this service is responsible for all the CRUD operations, i.e., creating, reading, updating and deleting, the *BB*, *BBI*, *Requirement*, *Project* and *Process* models. To execute these operations, the implemented REST API exposes, for each of these models, specific endpoints to create, read, update or delete executing POST, GET, PUT and DELETE requests.

4.3.2 Authentication Service

The *Authentication Service* implements the authentication and authorization flow of the prototype. This microservice is implemented in NodeJS with a gRPC communication mechanism. The motivation to have an authorization flow comes with the need of having different roles in the application. The *User* role has the permission to follow basic steps of the implemented method. The differences are that the *Admin* has the permission to create, update and remove the application models, and in the Process Creation Step, the *User* can only request the creation of the process. In contrast, the *Admin* can attend to the process creation request. Figure 4.5 shows the class diagram of the Authentication Microservice.

Figure 4.5: Authentication Microservice class diagram

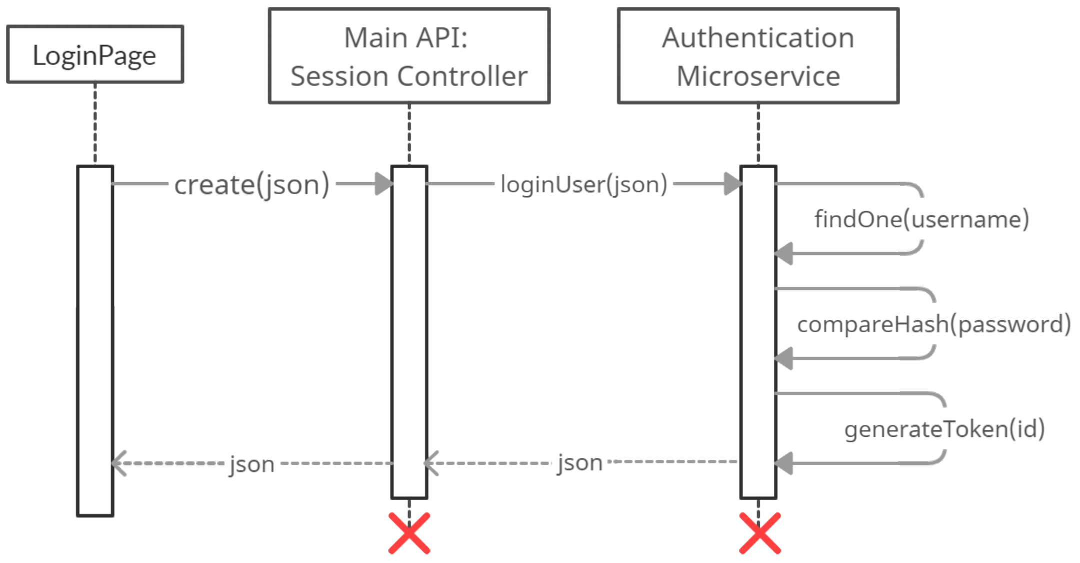


Source: The Authors

The decision to have a microservice module dedicated to the authorization flow is to decouple the need for authentication to the rest of the application's microservices. This way, the microservices can work alone without authentication, easing their development and making them detachable from the prototype. For example, the *Matching Microservice* can be easily used by external projects.

When the User authenticates in the prototype, they make a request to the Main API Microservice. The request is then forwarded to the Authentication Microservice where it is processed. In the body, the request contains the user's username and password. Once the request arrives, an entry on the User's database is searched containing the same username. If it is found, the password from the request's body is encrypted and then compared to the entry's password. If it is a match, a token is generated and returned to the Main API and forwarded to the request's source. This sequence is illustrate in Figure 4.6.

Figure 4.6: Authentication sequence flow



Source: The Authors

4.3.3 Matching Service

The Matching Service is responsible for recommending BBs and BBIs according to the selected Project Requirements. It is implemented using Python programming language and communicates with the Main API using gRPC, meaning both microservices share the same contract and know each other. Implementing the recommendation system into an individual microservice eases its use for different applications. It can be improved in future work, including developing a more complex recommendation system using artificial intelligence (RUSSELL; NORVIG, 2016). Moreover, the complexity of this feature can lead to heavy processing that can be handled by scaling the service, which the Microservice Architecture also aids.

According to Isinkaye et al. (2015), recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragments out of a large amount of dynamically generated information according to user's preferences, interests, or observed behavior about an item. In the case of the prototype, the recommendation system aims to reduce the overwhelming task of choosing among the massive heterogeneity of IoT components, filtering according to the user's requirements for a specific IoT environment configuration.

The recommendation system for the prototype matches the defined requirements with the BBs and BBIs of the toolbox. The match is performed first trying to fit a requirement to a BB's capability and then trying to match with a BBI's feature. If there is a match, the ID of the BB or BBI is appended to the corresponding match list.

4.3.4 BPMS Handler Service

The BPMS Handler Service is responsible for bridging the connection between the Main API and the desired BPMS. The prototype will be BPMS agnostic in future work, enabling the communication with several BPMS, e.g., Camunda, Signavio, Bonita. Currently, the service is capable of communicating with Camunda BPMS through Camunda REST API. This microservice is meant to implement all the necessary logic to create a process to the desired BPMS and manipulate it according to the chosen environment. The motivation to have a dedicated microservice to this responsibility is the expected complexity once different BPMS logic is added.

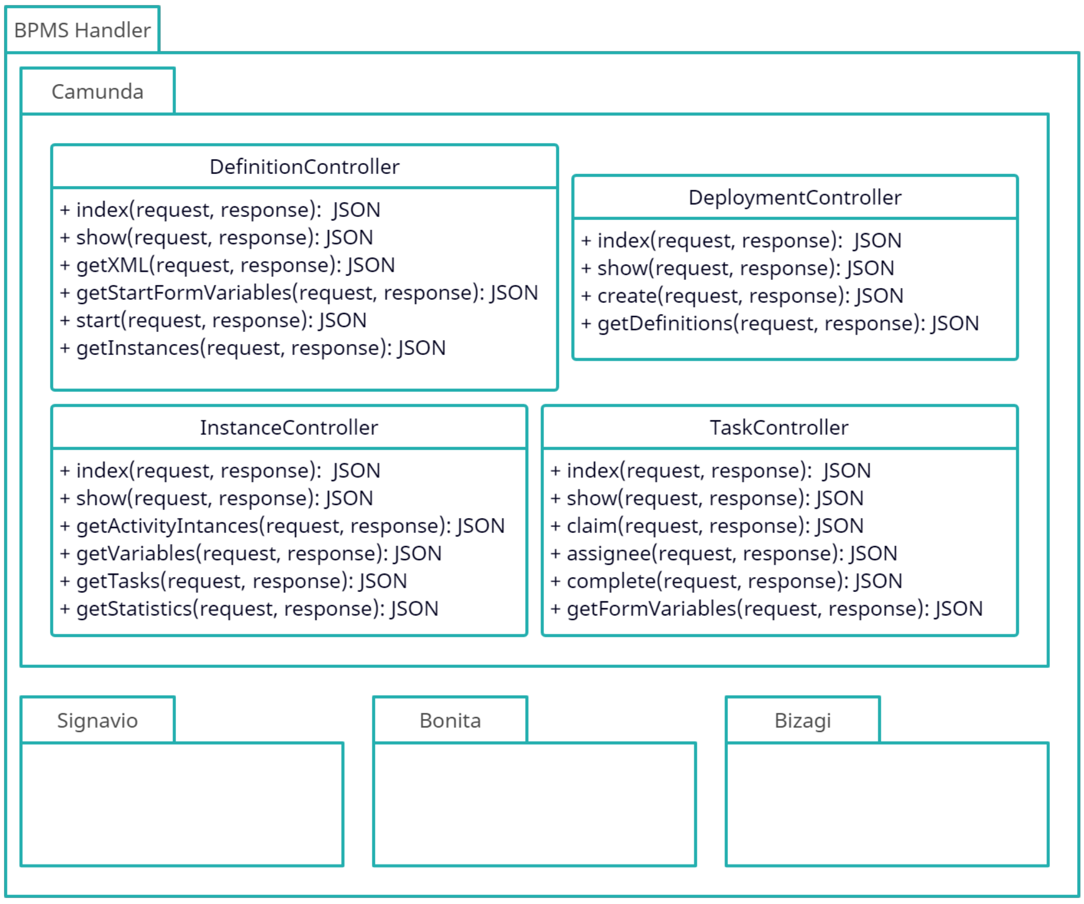
This microservice implements all the process execution environments, including creating a process deployment to the BPMS system by uploading a *.bpmn file*, creating a definition of the process on the BPMS, and starting an instance of the process definition, and complete tasks of the process instance. Figure 4.7 shows the class diagram of the microservice.

Camunda Platform

Camunda (CAMUNDA, 2021) is an open source platform from Germany, which supports the modeling and automation of business processes using BPMN, Case Management Model and Notation (CMMN) and Decision Model and Notation (DMN) diagrams. The Camunda's infrastructure and the typical user roles are depicted in Figure 4.8, the main components are described below:

- **Process Engine**: A Java library responsible for executing BPMN 2.0 processes. It has a lightweight core and uses a relational database for persistence.
- **Camunda Modeler**: The modeling environment for BPMN 2.0, CMMN and DMN diagrams.
- **Web applications**: Composed of a REST API – allows to use the process engine from a remote application or a JavaScript application, the Camunda Tasklist – a web application for human workflow management and user tasks that allows process

Figure 4.7: BPMS Handler class diagram


BPMS Handler
Camunda
DefinitionController
+ index(request, response): JSON
+ show(request, response): JSON
+ getXML(request, response): JSON
+ getStartFormVariables(request, response): JSON
+ start(request, response): JSON
+ getInstances(request, response): JSON
DeploymentController
+ index(request, response): JSON
+ show(request, response): JSON
+ create(request, response): JSON
+ getDefinitions(request, response): JSON
InstanceController
+ index(request, response): JSON
+ show(request, response): JSON
+ getActivityIntances(request, response): JSON
+ getVariables(request, response): JSON
+ getTasks(request, response): JSON
+ getStatistics(request, response): JSON
TaskController
+ index(request, response): JSON
+ show(request, response): JSON
+ claim(request, response): JSON
+ assignee(request, response): JSON
+ complete(request, response): JSON
+ getFormVariables(request, response): JSON
Signavio
Bonita
Bizagi

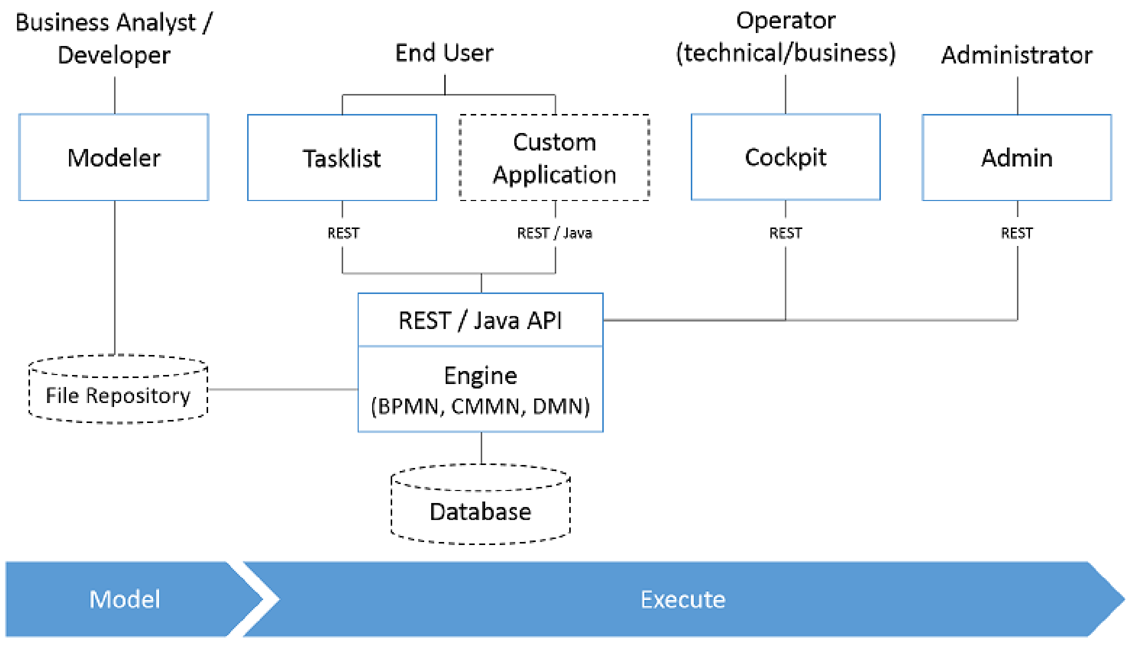
Source: The Authors

participants to inspect their workflow tasks and navigate to task forms in order to work on the tasks and prove data input, the Camunda Cockpit – a web application for process monitoring and operations, allowing to search for process instances, inspect their state and repair broken instances, and the Camunda Admin – a web application that allows managing users, groups and authorizations.

When deploying a set of BPMN 2.0 files to the process engine, a process deployment model is created, then it will also create a registration for this deployment with the process engine. If the deployment of an already deployed process is performed, it will be resolved based on the process definition keys. The user can start a process instance once the user has a process definition. Multiple instances can run in parallel, even from the corresponding process definition. Moreover, the process instances can have several tasks, where the User tasks can be inspected and manipulated through the Camunda Tasklist.

The BPMS handler service encapsulates all the logic needed for the prototype to communicate with the Camunda Platform through the Camunda REST API. This in-

Figure 4.8: Camunda's infraestructure and components



Source: Camunda Platform community (2021)

cludes the functionality of deploying a process uploading the corresponding BPMN 2.0 file to the Camunda process engine, starting an instance of a process definition, getting the respective process XML and task list, and also claim, assign, and complete user tasks.

4.3.5 Data Extractor Service

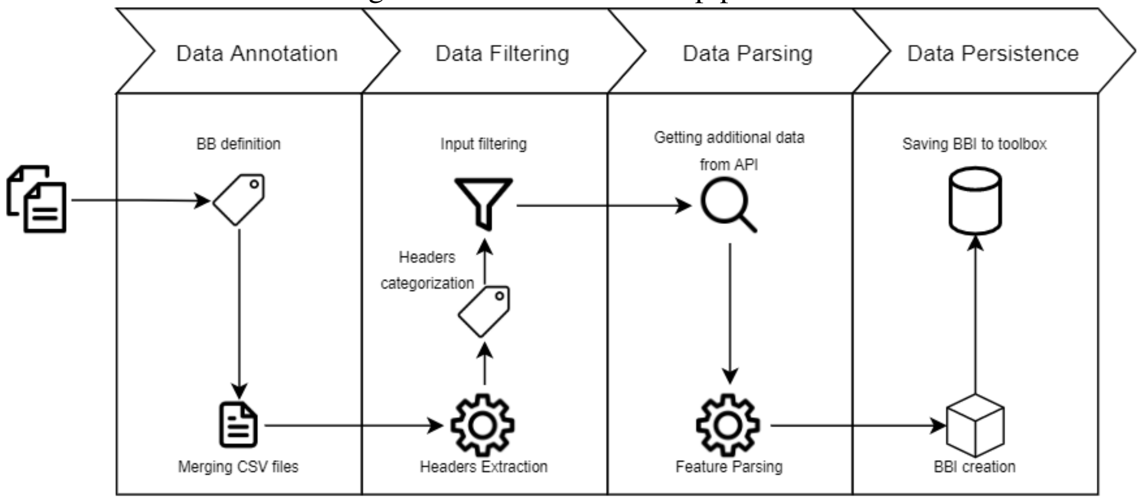
The Data Extractor Service is responsible for getting input data, usually .csv files, from well-known databases, filtering, aggregating, and parsing the data to create BBs and BBIs for the toolbox. Hundreds of BBIs were created this way so that the matching algorithm could be trained and for the toolbox to have a good range of options for user's applications.

The initial input batch data was from Mouser Electronics [2]. There are thousands of devices including sensors, actuators, microcontrollers, gateways, routers, wireless modules, and so on. Some categories of devices were selected and then their .csv files were used as input for the microservice. Each input data was annotated with a previously defined building block, e.g., csv files of Ambient Light Sensor downloaded from the Mouser website used as input for BBIs for the BB Ambient Light Sensor.

The data extraction pipeline is depicted in Figure 4.9. First, a batch of .csv files is

[2]https://www.mouser.com/

Figure 4.9: Data extraction pipeline



Source: The Authors

used as input. Then, the different files are merged into a unique file to ease the extraction. The second step of the pipeline reads all the headers from the individual CSV file. Then, these headers are annotated as mandatory and important. For example, Sensitivity and Acceleration can be annotated as mandatory, and Maximum and Minimum Operating Temperature can be annotated as important in a batch of Ambient Light Sensors files. With the list of mandatory files, the entries are then filtered. If the entry does not have all the mandatory headers, it is discarded, as is shown in Algorithm 1.

For each column of each filtered entry, if the value is valid, it is mapped to some feature. A BBI is composed of these features. Besides, more details are described for each entry with data from a request made to the Mouser API. After parsing the entry, the BBI is created with the information from the API plus the mapped features and is then saved to the toolbox database.

Algorithm 1: FILTERDATA filter an CSV file entry according to annotated headers

```
Input: An entry $e$ from a csv input file, and a set of $C = \{c_1, c_2, \ldots, c_n\}$
       columns with the respective annotaded header
Output: False for filtered and True for not filtered
1 $isValid \leftarrow true$
2 for $c \in C$ do
3     if $isHeaderValid(c)$ & $valueOf(c) = \emptyset$ then
4         $isValid \leftarrow false$
5 return $isValid$
```

4.4 Front-End

Frameworks and libraries can enhance JavaScript-based web applications to achieve complex visual features and maintain a clean code structure overall. Therefore, to implement the Frontend, we collectively chose the JavaScript programming language with the ReactJS [3] library, an open-source JavaScript library for building user interfaces and UI components for web and mobile applications.

According to Vipul and Sonpatki (2016), ReactJS tries to solve the problem from the View layer. It can very well be defined and used as the V in any of the MVC frameworks. It breaks down parts of the view in the Components. These components encompass both the logic to handle the display of view and the view itself. In addition, it can contain data that it uses to render the state of the app.

Furthermore, the ReactJS library is used for building modular user interfaces using a Single Page Application (SPA). The SPA principle enables web applications that do not require a complete reload of the page to perform a change in the display or perform a user interaction (VIPUL; SONPATKI, 2016). Moreover, Aggarwal (2018) states that ReactJS shows high performance in comparison to other widely used frameworks and libraries mainly due to the modifications performed first on a virtual Document Object Model (DOM), followed by an alignment with the browser's DOM, as opposed to the standard approach of directly modifying the browser's DOM.

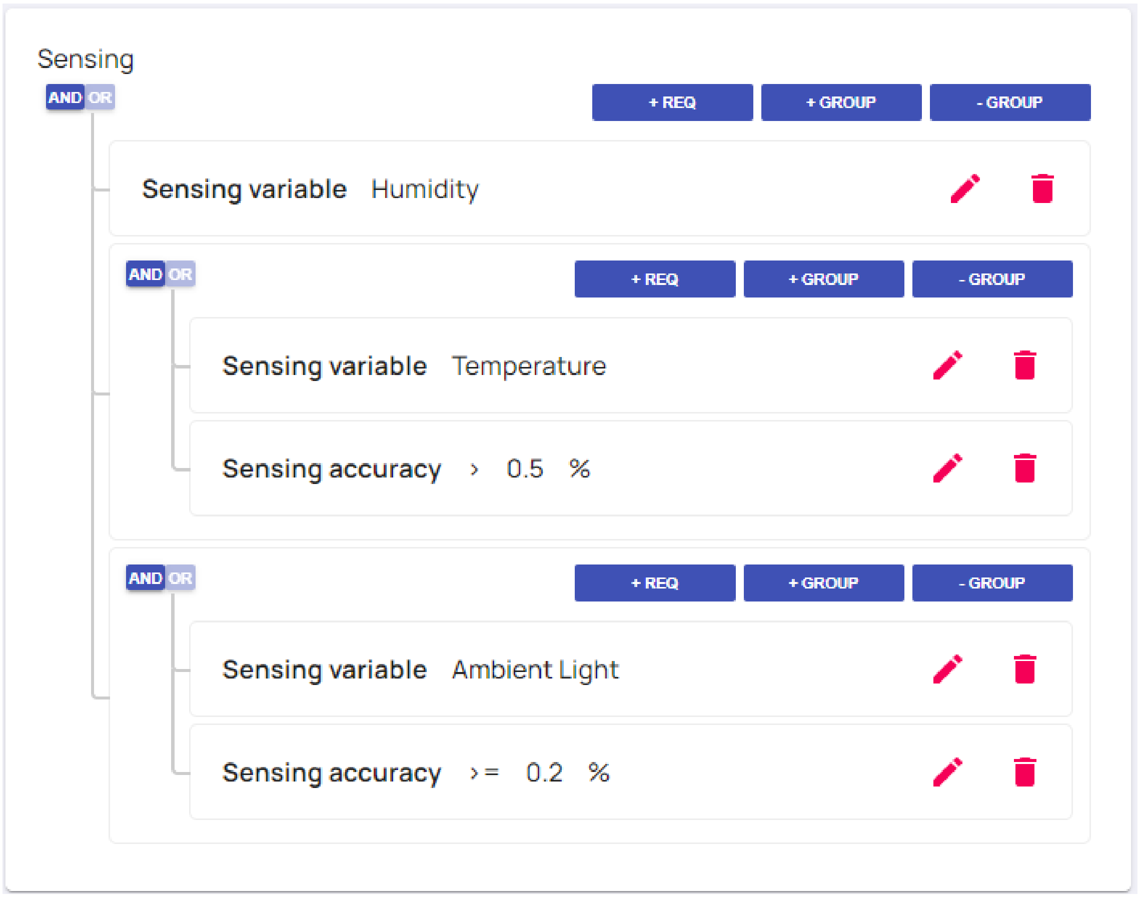
The usage of ReactJS enables the integration with several modular components for visualization and interaction of features. It leverages all the development and implementation of the Toolbox and the holistic method by allowing the implementation of one or more Components to a Toolbox definition (e.g., BB) or, for example, a Step of the method. Therefore, the Front-End represents a platform containing the Toolbox, including an interface for the *Requirements*, *Building Blocks*, and *Building Blocks Implementations*, and also contains the visualization and execution of each step of our holistic method.

Figure 4.10 illustrates the selected requirements for a specific project. In this case, the requirements for the *Sensing* category are Sensing of Humidity AND Sensing of Temperature with at least 0.5% of accuracy AND Sensing of Ambient Light with at least 0.2% of accuracy. Moreover, Figures 4.11 and 4.12 illustrate an example of BB and BBI components interface, respectively.

Finally, an important component in terms of visualization is the Stepper, shown in

[3]https://reactjs.org/

Figure 4.10: RequirementsPages component visual interface


Sensing
AND OR
+ REQ
+ GROUP
- GROUP
Sensing variable Humidity
AND OR
+ REQ
+ GROUP
- GROUP
Sensing variable Temperature
Sensing accuracy > 0.5 %
AND OR
+ REQ
+ GROUP
- GROUP
Sensing variable Ambient Light
Sensing accuracy >= 0.2 %

Source: The Authors

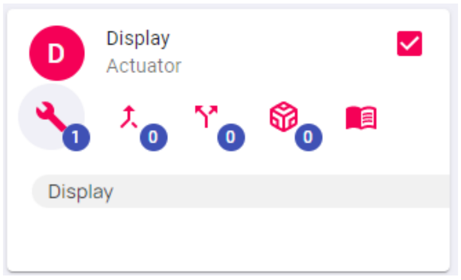
Figure 4.13. For the end-user, it represents which is the current step of the method that the application is currently in. The colored steps are already finished and the grey colored steps are not yet executed.

In the context of the developed prototype, the chosen web architecture, programming language, and library enabled a straightforward usage of external components and libraries. ReactJS provides features which are particularly suitable for the implementation of the platform overall: a component-based software engineering (CBSE).

CBSE allows the development of independent pieces of functionality as entities that can be composed and when defined with clear interfaces, it promotes reusability, as is the case with third-party external components (HOVLAND et al., 2003), and an effortless integration with Cascading Style Sheets (CSS), external components (e.g., entire design systems), and other libraries (e.g., bpmn.io).

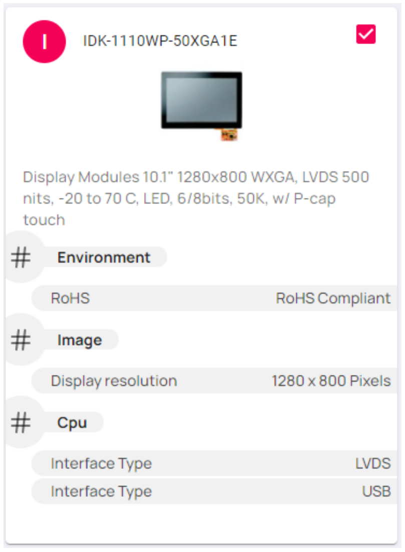
Fedosejev (2015) details the typical architecture and primary component relations. According to him, the App component is the entry point of a ReactJS application. Furthermore, the App component is the orchestrator of the application and is responsible for

Figure 4.11: Example of a BB component interface



Source: The Authors

Figure 4.12: Example of a BBI component interface



I IDK-1110WP-50XGA1E

Display Modules 10.1" 1280x800 WXGA, LVDS 500 nits, -20 to 70 C, LED, 6/8bits, 50K, w/ P-cap touch

Environment

RoHS	RoHS Compliant

Image

Display resolution	1280 x 800 Pixels

Cpu

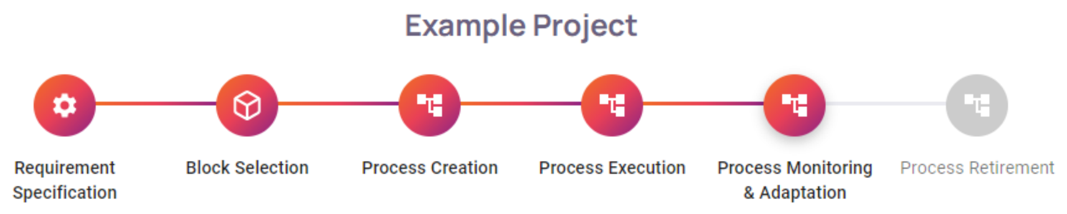
Interface Type	LVDS
Interface Type	USB

Source: The Authors

instantiating the smaller modular components, i.e., the components designed to be the interface of the Toolbox and the method steps. Figure 4.14 shows the class diagram of the designed components, and a brief description of each one is listed below:

- *LoginPage*: This component is responsible for authenticating the end-user in the system. The user, after login, is redirected to the *Dashboard* component.
- *Dashboard*: The *Dashboard* is the home component of the application. In this component, it is possible to see the number of relevant features of the prototype, e.g., BBs, BBIs, and Projects. Also, this component has a button to initiate the process and redirect the user to the *ProjectPage*.
- *ProjectPage*: This component is responsible for setting the current project of the end-user. The user can also create a new project, describe its name, or select a previously created project. After selecting or creating a project, the user is redirected to the *RequirementsPage*.
- *RequirementsPage*: The current component corresponds to the first step of the

Figure 4.13: Stepper component



Source: The Authors

method. While in this component, the users can select and combine a set of pre-defined requirements for their IoT environment with an easy-to-use interface.

- *BBPage*: At the current component, the users can select the suitable BB for their IoT environment. It is shown for the user a list of BBs registered in the Toolbox. According to the matching algorithm, the recommended BBs are highlighted to help the user choose the BBs that fit its IoT environment requirements.
- *BBIPage*: Each BB has one or more BBIs shown on the *BBIPage*. The listed BBIs in this component correspond to a BB. Moreover, to get to this page, the user must interact with a certain button of a BB on the *BBPage*.
- *ProcessCreationPage*: This component has different visualizations according to the user profile. For the common user, in this component, after selecting the BBs and BBIs, the user requests the creation of a process to set up the desired IoT environment. After the process is available, the user can then be redirected to the ProcessExecutionPage. On the other hand, for a user with the **admin** profile, in this component, it is possible to upload a process for creating a previous request and, additionally, to upload a process for the retirement of the respective IoT environment.
- *ProcessExecutionPage*: The current component is responsible for the execution of a given process by executing a corresponded *bpmn* file. It makes use of external libraries (bpmn.io[4]) to draw an interface for interacting with the process. Also, the completion of the process tasks is made with the communication with some BPMS systems. This component acts for the process creation as well as for the process retirement.
- *ProcessMonitoringPage*: In the current state, this component is responsible for acting as the feedback loop through the method. In future work, it will be possible to integrate with some IoT platforms to monitor the environment.

[4]https://bpmn.io/

Whenever an end-user is redirect to a different page component, the application triggers a request to the Back-End 4.3, waits for a successful response with metadata and data received through a JSON format, and loads the respective content to the page, e.g., a list of BBs.

The Front-End supports a number of possible sequence flows, depending on which action the end-user performs on the web application. Figure 4.15 illustrates a use case of selecting requirements for a user project. At the initial point of the diagram, the App component is at the *LoginPage* component, the *login* method is called in order to authenticate and the request is properly handled by the Backend. Once the response is received, the user is authenticated, and the received metadata payload is stored, then the user is redirected to the *Dashboard* component. The user can click on the floating bottom to call the *goToProjectsPage* method and is then redirect to the *ProjectPage* component. There, they can select a project through the *handleSelectProject* method, which is called once the user clicks on the respective button. After selecting a project, the user is redirected to the *RequirementsPage*. Finally, depending on the user's IoT environments, they can select different requirements with the method *handleAddRequirement*, remove a requirement with the *handleDeleteRequirement* method, or add and remove requirement groups with *handleAddGroup* and *handleDeleteGroup*, respectively. After selecting all the requirements, the user can then use the *handleNextStep* method to go to *BBPage*.

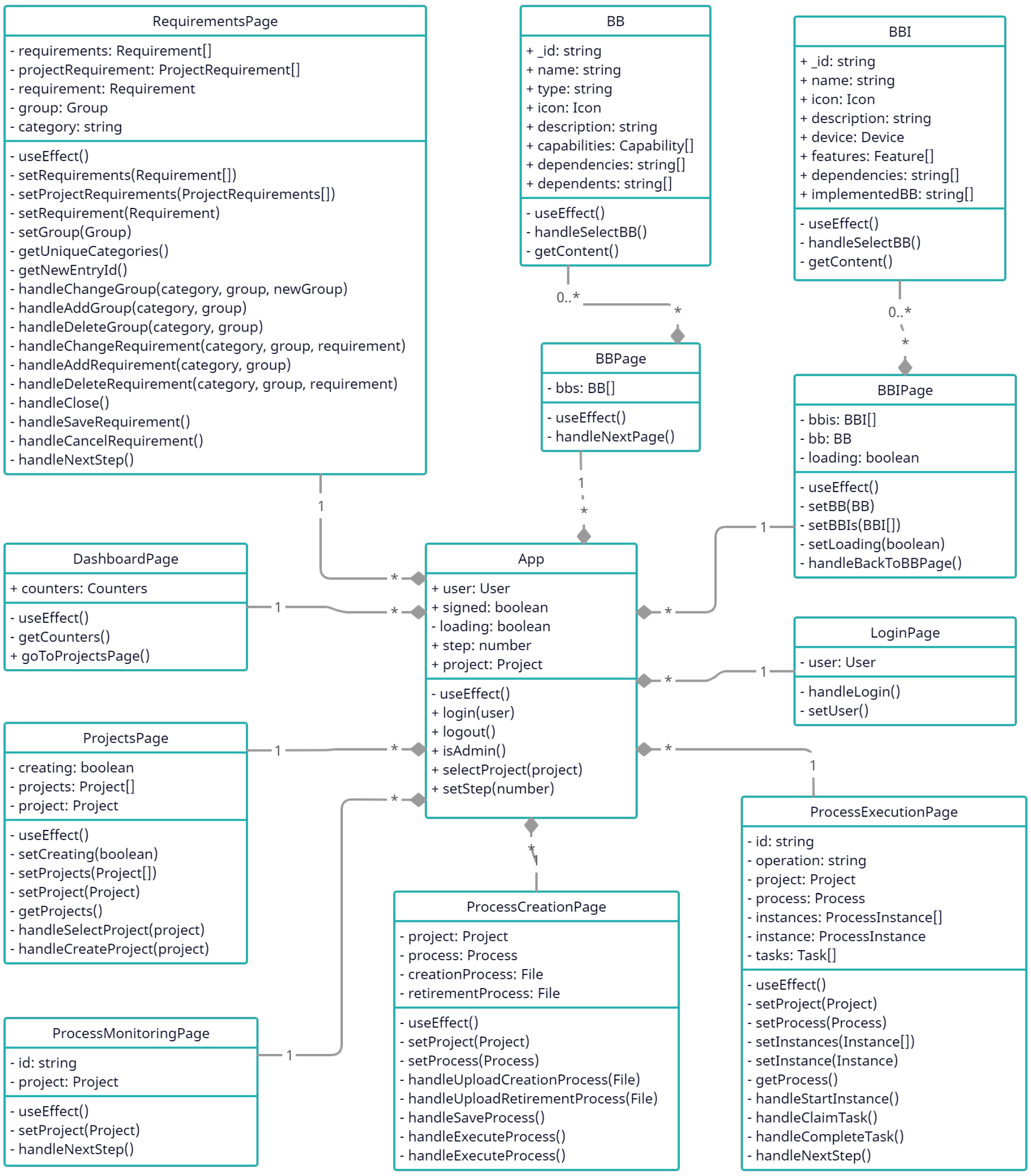
4.5 Chapter Summary

In this chapter, first, we presented the elicited requirements for choosing the prototype architecture. Therefore, we decided on Microservice Architecture because it offers modularity, reusability, and scalability. Besides, we presented the main definitions and challenges developing using this architecture. The second section showed the technologies used throughout the project's development, including JavaScript, Python, Docker, MongoDB, REST API, and gRPC.

After presenting the technologies, the server component and its microservices were described in detail, including the main API, responsible for the creation, reading, updating, and removal operations of the main application models; the authentication service, responsible for implementing the authentication and authorization flow; the matching service, which implements the matching algorithm between requirements and BBs and BBIs; the BPMS service, designed to encapsulate the logic of interaction between different BPMS; and the data service, which is responsible for extracting data and creating BBs and BBIs from them.

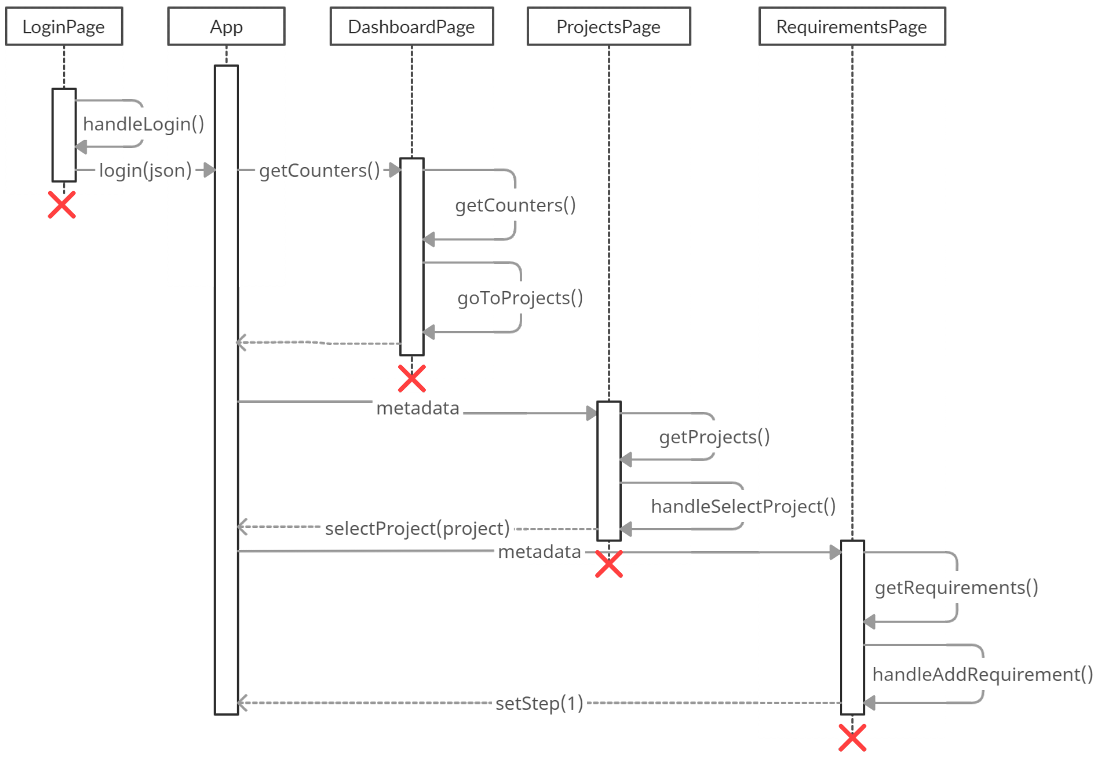
Section 4.4 presented the client-side of the application, containing details about the implementation of the prototype interface and its technologies used and demonstrating with class and sequence diagrams the respective models and interactions.

Figure 4.14: Class diagram of the Front-End



Source: The Authors

Figure 4.15: Sequence Flow example of the Front-End



Source: The Authors

5 EVALUATION

In this chapter, we evaluate and discuss our work. The evaluation of the approach presented in this study was performed by demonstrating the applicability of the approach, using the prototype, to a real-world case study. Furthermore, this chapter also addresses strengths and weaknesses and possible improvements, besides what is expected for future work.

It is necessary to recall the goals and research question of the work to assess the results of the approach and the prototype developed. The general goal of this work is to provide a holistic method that describes the necessary steps domain experts need to take to set up IoT systems from scratch. The Research Questions are listed below:

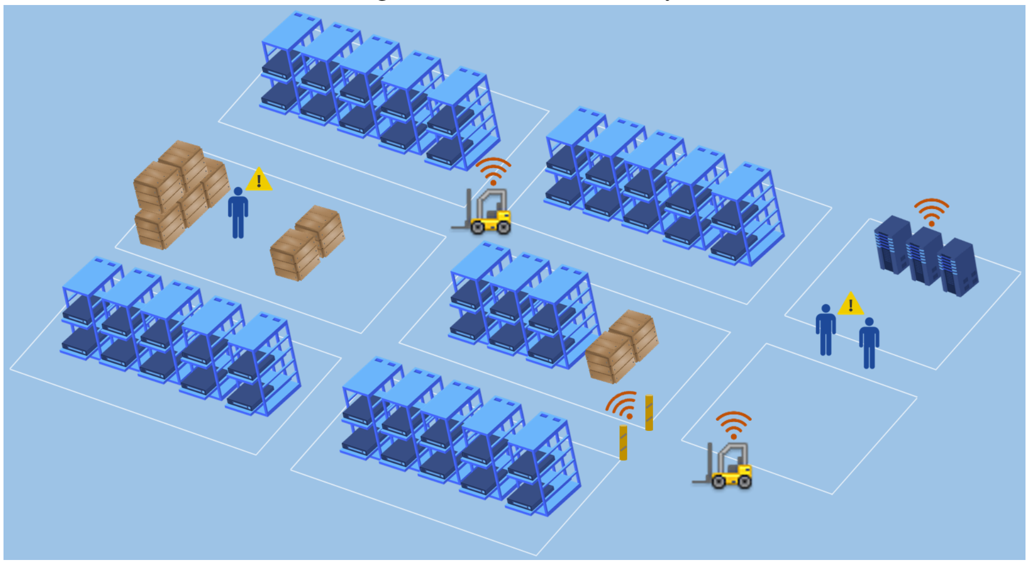
- **[RQ1]**: Is it possible to represent the considerable diversity of components of the IoT domain using Building Blocks?
- **[RQ2]**: Can IoT Building Blocks help in the IoT-aware process modeling?

5.1 Case Study

This section presents a case study that applies our approach to a Smart Factory scenario. In this scenario, depicted in Figure 5.1, the indoor localization of self-driving vehicles should be realized. For example, these vehicles can pick up materials in a warehouse and transport them to a different location. In order to realize this scenario, so it is robust enough for real-world use, it must be known where the vehicles are at all times. Each vehicle needs a localization tag, which communicates with multiple stationary tags throughout the factory to determine the current location. Based on this data, an edge cloud system calculates the position and determines the paths of the vehicles.

Furthermore, electric doors separate different factory parts and enable access to the outdoors, e.g., to load goods into trucks. These doors need to open automatically as soon as a vehicle approaches them. To make the scenario even more complex, people are also moving around in the factory and need to be observed by multiple vehicles' sensors. Hence, many sensors and actuators are required to be set up. This scenario is very complex and can show the benefits of our toolbox. In the following, the steps of our method are described specifically for this context expecting the configuration of the indoor localization system.

Figure 5.1: Smart Factory



Source: The Authors

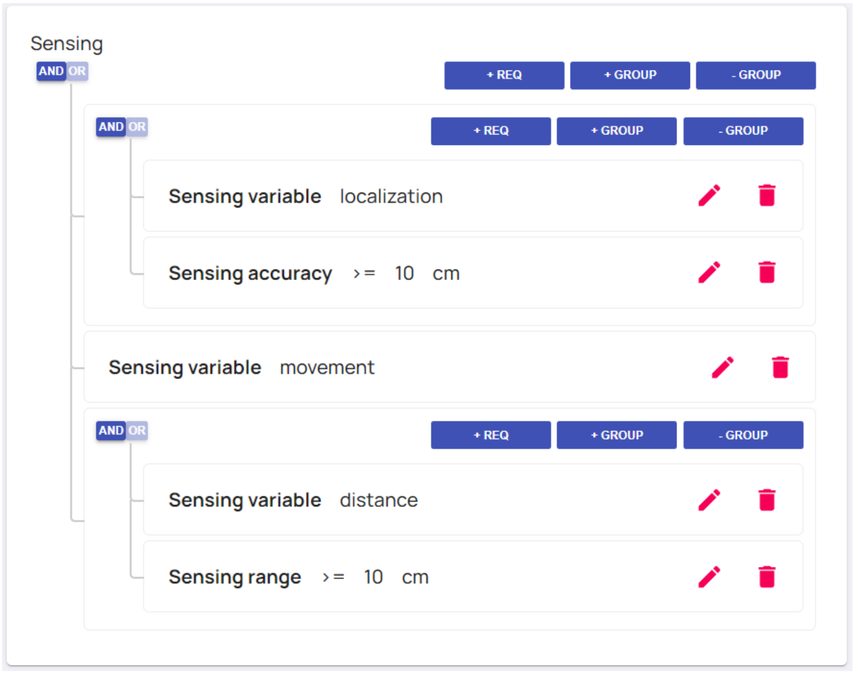
Step 1: Requirement Specification

The requirement specification for this use case requires many different stakeholders: Shop floor workers who have experience with the scenario's environment, network experts - setting up wireless connectivity and the edge cloud environment; indoor localization experts - setting up the localization system; electricians - making sure that the power supply for the localization is guaranteed; business experts - bringing in knowledge about the processes, and so on. In a workshop, these stakeholders need to get together and define the requirements for the scenario. These requirements include functional ones, e.g., an indoor localization system, person detection, door control, and non-functional requirements, including safety, security, privacy, and efficiency and accuracy of the localization and robustness.

The toolbox provides a set of requirements where all the requirements mentioned above can be select to be further used by the matching system to recommend building blocks accordingly. An example of selection of the requirements provided by the toolbox for the *sensing* category is depicted in Figure 5.2.

There are many possibilities of requirements combination using the pre-defined requirements of the prototype. The user can select requirements and the relation of each group or requirement individually. For example, possible selections for the *communication* category is depicted in Figure 5.3, and, in the first group, we can observe that the OR condition is selected, meaning the matching algorithm will search for any component that

Figure 5.2: Requirements selections for the sensing category for the case study indoor localization system



Sensing variable localization

Sensing accuracy >= 10 cm

Sensing variable movement

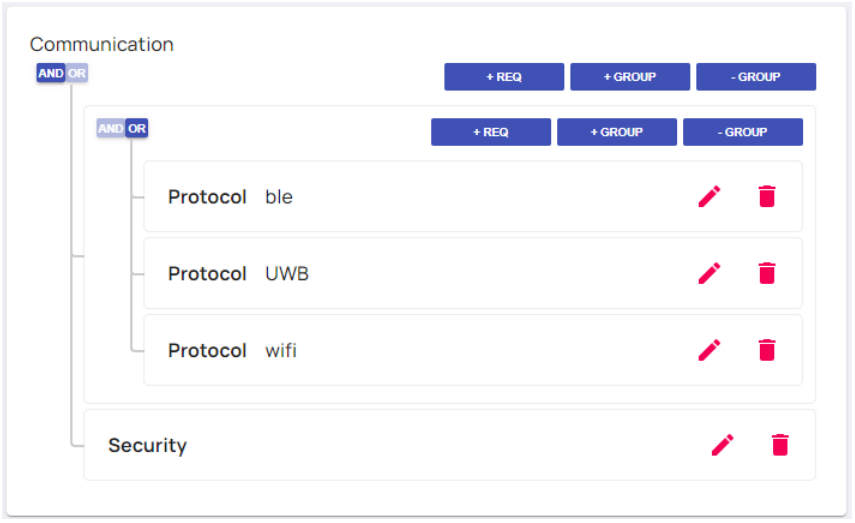
Sensing variable distance

Sensing range >= 10 cm

Source: The Authors

uses BLE, UWB, or WiFi communication protocol. Moreover, the outer group will filter only for the BBs and BBIs that also match the *Security* requirement.

Figure 5.3: Requirements selection for the communication category for the case study indoor localization system



Protocol ble

Protocol UWB

Protocol wifi

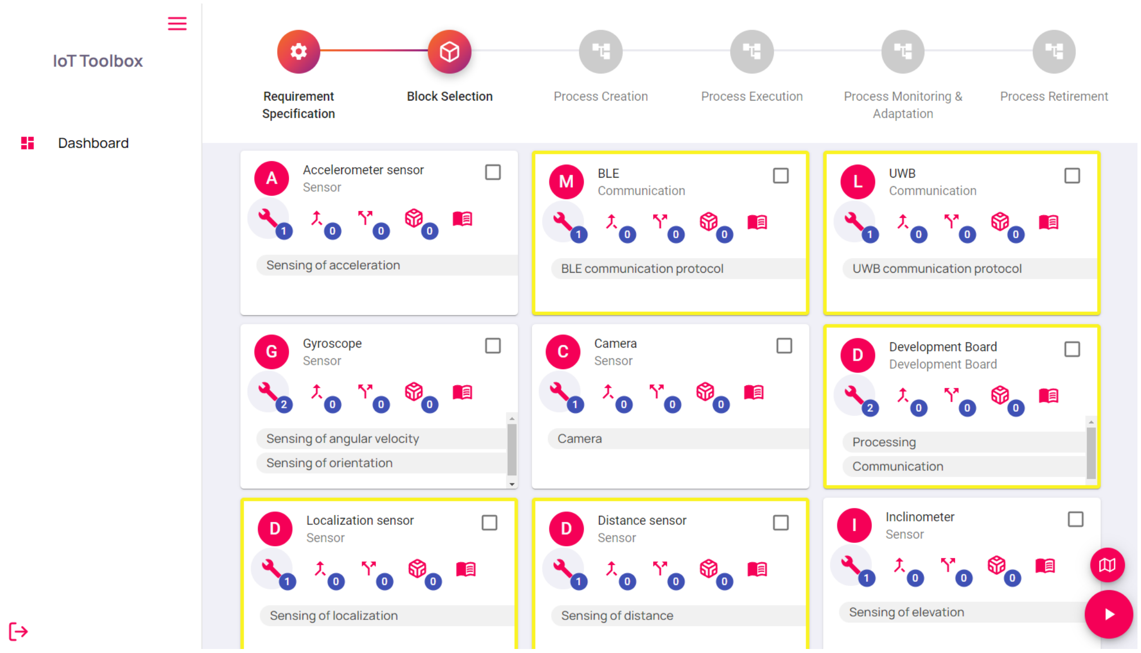
Security

Source: The Authors

Step 2: Building Block Selection

After the requirements are selected, the corresponding BBs and BBIs are recommended in the Building Block selection step and can be selected by the stakeholder. For example, one requirement of the indoor localization system is the *Sensing of localization*. The toolbox can propose using BLE localization through triangulation or more exact systems, using a real-time localization system, e.g., based on ultra-wideband (UWB). Since the requirement *Sensing Accuracy* is also given, the recommendation suggests a more accurate UWB based system. Furthermore, wireless communication might be required. Due to the high efficiency requirements, a 5G network might be suggested instead of WiFi or LTE.

Figure 5.4: List of prototype's BBs with recommendations for the case study.

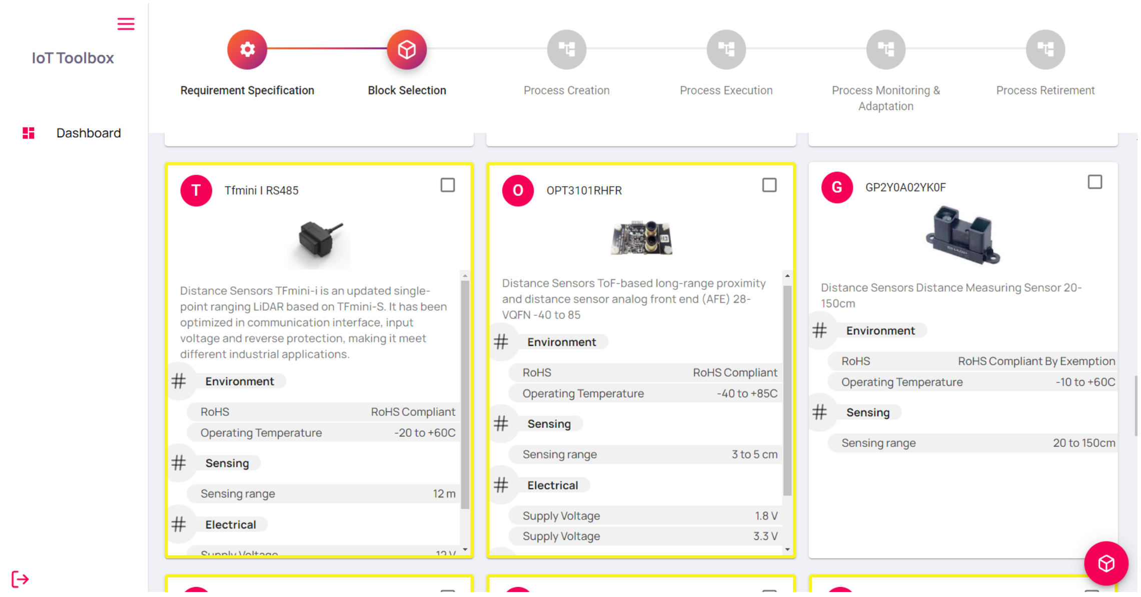


Source: The Authors

The suggestions of the toolbox are an essential part to decrease the time necessary to analyze and select the suitable components for the scenario, which usually can take months. However, if the domain experts prefer a not recommended Building Block, it can be selected instead with no problem to the further creation process. Figure 5.4 illustrates some pre-defined BBs of the toolbox, where the recommended BBs are highlighted.

The matching system is executed right after the requirements are selected, and the user goes to the Building Block page. The specified requirements are the input for the matching algorithm (see 4.3.3). Once the algorithm finishes, a list with the BBs and BBIs is returned that matches the requirements. This way, the user can see in the platform the

Figure 5.5: Prototype's BBIs of the Distance Sensor BB with recommendations for the case study



Tfmini I RS485

Distance Sensors TFmini-i is an updated single-point ranging LiDAR based on TFmini-S. It has been optimized in communication interface, input voltage and reverse protection, making it meet different industrial applications.

Environment

RoHS	RoHS Compliant
Operating Temperature	-20 to +60C

Sensing

Sensing range	12 m

Electrical

OPT3101RHFR

Distance Sensors ToF-based long-range proximity and distance sensor analog front end (AFE) 28-VQFN -40 to 85

Environment

RoHS	RoHS Compliant
Operating Temperature	-40 to +85C

Sensing

Sensing range	3 to 5 cm

Electrical

Supply Voltage	1.8 V
Supply Voltage	3.3 V

GP2Y0A02YK0F

Distance Sensors Distance Measuring Sensor 20-150cm

Environment

RoHS	RoHS Compliant By Exemption
Operating Temperature	-10 to +60C

Sensing

Sensing range	20 to 150cm

Source: The Authors

corresponding matches as it is highlighted. To illustrate this scenario, Figure 5.5 shows possible matches for the *Distance sensor* BBI for this context.

Step 3: Process Creation

Once the stakeholders select the requirements, BBs and BBIs, they can request the process creation to configure the desired environment. The process creation is conducted by an IoT expert and is, for now, designed manually. All the selected BBs and BBIs are saved as a project in the toolbox, and then a specialist can handle it and create a process accordingly. Moreover, best practices based on previous scenarios can help in process creation.

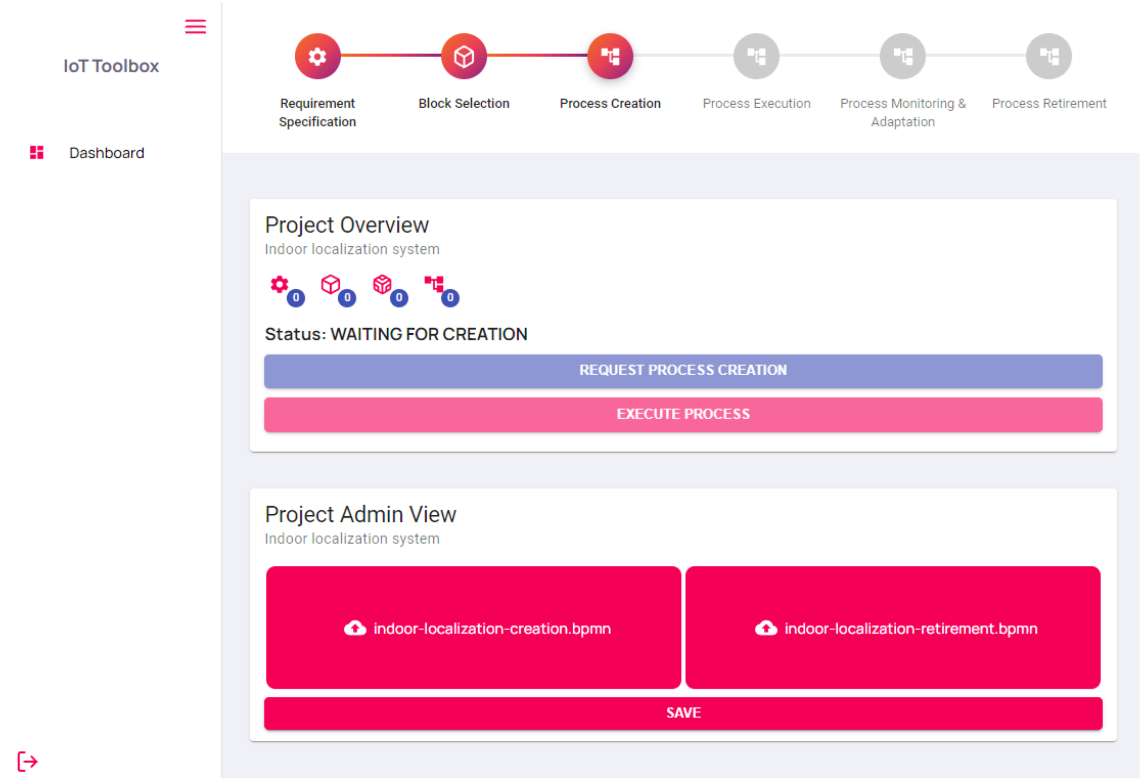
For complex scenarios, such as an entire floor configuration of a Smart Factory, multiple processes are required to set up the specific parts (e.g., localization, door control, networks, etc.). The process regarding the selection in previous step was created as shown in Figure 3.4 for the localization system.

The IoT expert models the respective process considering the selections, and the modeling can be made in any software capable of modeling BPMN 2.0 processes, e.g., Camunda Modeler[1]. Once the IoT expert has the *.bpmn* file of the modeled process, the request can be attended and the *bpmn* file is uploaded to the platform. Finally, the

[1]https://camunda.com/products/camunda-platform/modeler/

stakeholder can execute the modeled process.

Figure 5.6: Prototype interface for process creation step


IoT Toolbox
Dashboard
Requirement Specification
Block Selection
Process Creation
Process Execution
Process Monitoring & Adaptation
Process Retirement
Project Overview
Indoor localization system
Status: WAITING FOR CREATION
REQUEST PROCESS CREATION
EXECUTE PROCESS
Project Admin View
Indoor localization system
indoor-localization-creation.bpmn
indoor-localization-retirement.bpmn
SAVE

Source: The Authors

Figure 5.6 shows the interface of the Process Creation step on the Admin, i.e., IoT expert, view. In our approach, a process retirement model might be uploaded together with the process creation model as a best practice using the prototype. It can be made interacting with the respective `Upload Button`.

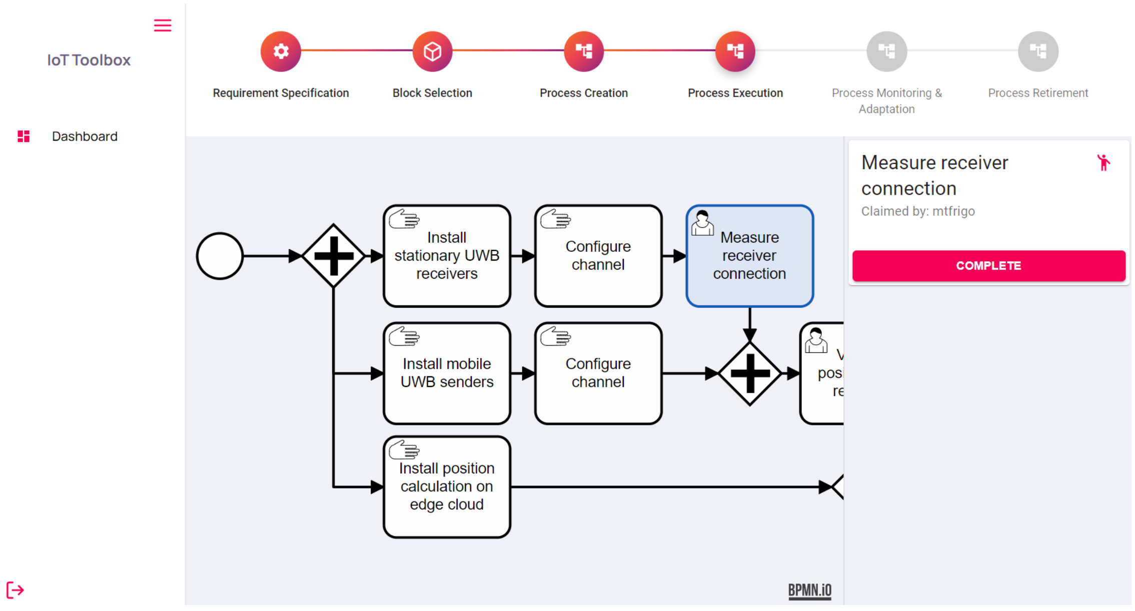
Step 4: Process Execution

In order to set up the desired scenario, the process in Figure 3.4 is executed. The corresponding experts install the hardware, e.g., the localization system, by using the steps given by the process. In the case of user tasks, the user should fill the variable(s) of the corresponding task(s) if it is the case.

The prototype provides an interface that shows the current process instance and the tasks. Figure 5.7 shows the interface. On the left of the figure, the user can visualize the big picture of the process, while on the right, the tasks are listed correspondingly.

After all tasks and processes have been executed, the scenario is set up. In general, during process execution, there might be dependencies between the processes. For exam-

Figure 5.7: Prototype interface for process execution step


IoT Toolbox
Dashboard
Requirement Specification
Block Selection
Process Creation
Process Execution
Process Monitoring & Adaptation
Process Retirement
Install stationary UWB receivers
Configure channel
Measure receiver connection
Install mobile UWB senders
Configure channel
Install position calculation on edge cloud
BPMN.io
Measure receiver connection
Claimed by: mtfrigo
COMPLETE

Source: The Authors

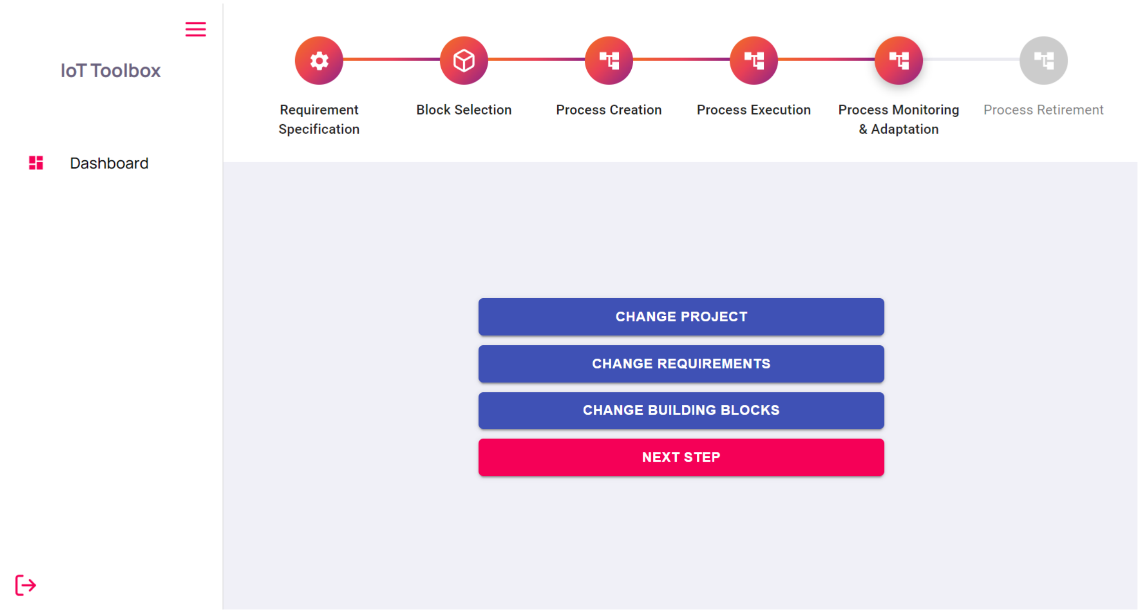
ple, it is necessary to set up wireless communication first before the localization system can be set up. This needs to be modeled in the process.

Step 5: Process Monitoring and Adaptation

In step 5, the stakeholders could adapt the IoT environment, for example, to extend the area the self-driving vehicles can move into or add more vehicles. This adaptation usually requires installing more UWB receivers on the walls or UWB senders on the vehicles. The overall setup should stay the same. Only necessary extensions should be made in the IoT environment. In this case scenario, an *adaptation process* includes, e.g., setting up more UWB senders and receivers and an additional configuration and integration step.

Figure 5.8 shows the prototype's interface for this step. The *adaptation process* can be requested once the user reaches the *Process Creation Step* again, after changing requirements, BBs or BBIs. In future work, in this step's interface, the corresponding Smart Environment variables might be visible by integrating the prototype with some IoT platform. This way, the process monitoring would be possible.

Figure 5.8: Prototype interface for process monitoring and adaptation step



Source: The Authors

Step 6: Process Retirement

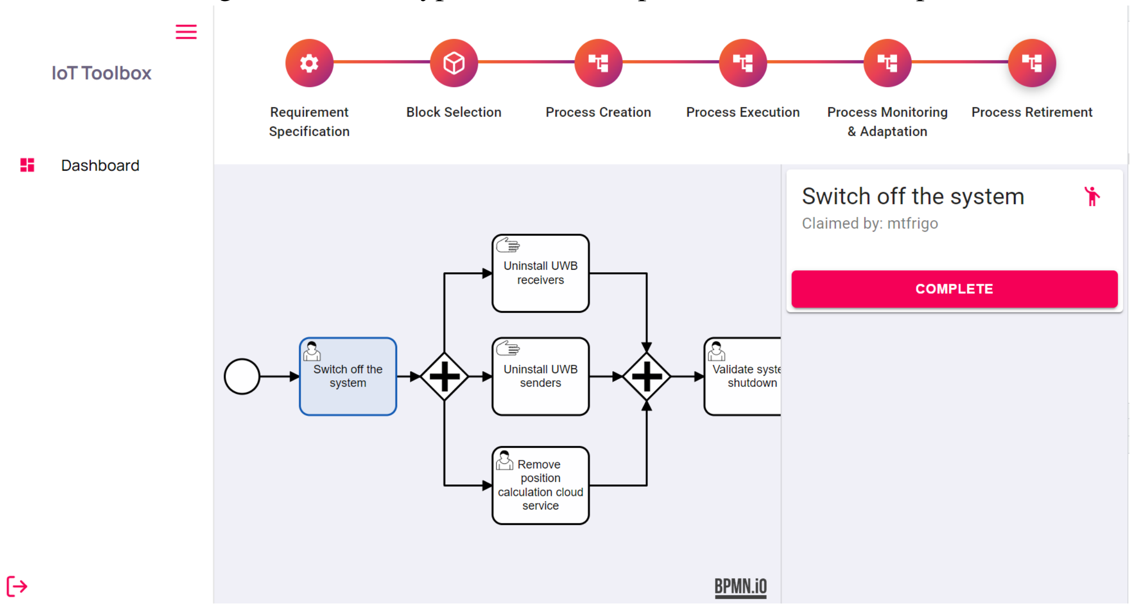
Possible cases when such a scenario is retired are moving the factory to another location or using other kinds of devices that fundamentally change the setup. In these cases, a *termination process* reverses all steps of the setup, for example, by removing the UWB senders, powering down the edge cloud, etc. The Figure 5.9 illustrates this scenario. The *termination process* might be provided together with the *creation process* in Step 3, although, running IoT environments might have changes systematically, e.g., a component might be missing. In this case, a new *termination process* must be designed and provided by the IoT expert.

5.2 Discussion and Future Work

In this chapter, we performed the evaluation of the approach through the case study applied to the prototype, for which the main objective was to configure an indoor localization system for a Smart Factory scenario. As such, the applicability of our holistic method to a real-world situation revealed many positive points, challenges for future work of our method and enhancements of the prototype. Moreover, this experiment leads us to answer our Research Questions.

The *Requirements Specification* is a crucial step for the whole method to operate

Figure 5.9: Prototype interface for process retirement step



Source: The Authors

accurately, and it is highly dependent on the stakeholders engagement of the project, meaning a bad specification of requirements can compromise the succeeding steps. The prototype supplies a pre-defined set of requirements, which can restrict more complex projects or simple projects where the requirements are not found on the prototype. On the other hand, as a positive aspect, the built prototype provides means of relation among the selected individual requirements and group of requirements, i.e., OR and AND conditions, as shown in the blue buttons in Figure 4.10. This way, the user can establish different combinations, allowing a broader recommendation for BBs and BBIs.

In between Step 1 and Step 2, the Matching Algorithm takes place. The algorithm is critical for one of the main purposes of this work: reducing the time stakeholders spend to deal with the wide heterogeneity of the IoT devices. The developed algorithm is a key feature for future work, whereas a more robust recommender system can be implemented, for example, using techniques such as artifical intelligence. The Matching Service implemented for the prototype aimed for matching the requirements selected in Step 1 and the registered BBs and BBIs of the Toolbox. For the propose of this work the service worked fine, but it is evident that is highly dependent of the selection of requirements and the properly modeling and creation of requirements, BBs and BBIs.

The *Building Blocks Selection* step aims for helping the stakeholders to select the IoT components for their IoT environments providing an abstraction of the common IoT devices and technologies (e.g., sensors, actuators, Wi-Fi, BLE, etc.) in terms of what we named on our approach of BB and its implementations BBI. In fact, it was possible to create this abstraction to many IoT components following our formalization, which answers our first research question positively. However, an extensible map study and comparison of other means of abstracting IoT components need to be realized for future work. Moreover, developing ways to provide artifacts and interfaces besides the description of BBIs, which can support automation for provisioning and configuration of the IoT environments, are also pursued for future work.

In the *Process Creation* step, the selection of the BBs and BBIs guides the business process modeling, and its creation needs to be conducted manually by experts. Notwithstanding, creating the business process can be time-consuming and overwhelming, and our approach does not provide any means of automation yet. Moreover, in the current state of the project, the BBs and BBIs help the IoT expert to know the desired project's components, but not its expected operation. In order to integrate our BBs and BBIs with the smart environments in a more efficient and robust way, the mentioned artifacts and interfaces would provide more extensive help for the configuration of the smart environments and the execution of the IoT-aware processes.

Thereby, our BBs and BBIs have considerable potential to help in the IoT-aware process modeling, although this approach must be extensible for more real-world cases to affirm its meaningful guidance.

The *Process Execution* and *Process Retirement* steps execute the business process designed to create and retire the IoT environment, respectively. Both implemented components share the same Components (Front-End) and the same services (Back-End), where the difference in operation is the inserted *bpmn* file. The integration of the well-known Camunda BPMS reinforces the potential applicability of the developed prototype. Furthermore, different BPMS should be integrated into future work, as an individual microservice encapsulates the integration logic, which imposes this extensibility.

The *Process Monitoring and Adaptation* step is responsible for monitoring once the IoT environment is configured. There are several commercial and open-source IoT platforms for the analysis and monitoring of IoT environments. Thus, in future work, the integration of the prototype with an IoT platform could enhance the approach for the whole lifecycle of smart environments through the developed prototype. Also, in this step is possible to change the requirements, BBs, or BBIs of the application allowing the adaptation of the IoT environments, not needing to retire it, but adapting through an adaptation process.

The application of our method to a hypothetically real-world scenario raised some points. First, because in some cases the technical specification of the devices is scarce, or presented in different formats. Finally, heterogeneity in components, whether in the same or different categories, has a complex model as a consequence.

The other point addresses the struggle to provide the quantitative measure of non-function requirements, such as security, energy, robustness, privacy, and many others. According to Venčkauskas et al. (2016), security, for example, at the communication level, is evaluated by the communication protocols, cryptographic algorithms, key management protocols, attack detections and preventions, secure routing, secure location detections, secure data distributions, and other mechanisms. The energy at this level is evaluated by routing algorithms, the use of sleep modes, and other mechanisms. This hampers the development of a robust algorithm in order to match the requirements with BB capabilities or BBI features and find the more suitable selection for the configuration of the IoT environment.

5.3 Chapter Summary

This chapter details the evaluation of our method in terms of the applicability of our holistic method to a hypothetically real-world case study. We first introduce the case study and then addresses the motivation to use our method to cope with the complexity and reduce the time of development of the configuration of this smart environment. Then, we detail a Smart Factory case study where an indoor localization system is set up. Furthermore, the application of our method is executed through the developed prototype.

Each step of our method applied to the smart environment configuration is described in detail, which the positive and negative points are raised for each step, for the method, and the prototype. We also present expected future work to solve the negative issues. Moreover, some overall challenges and difficulties regarding our approach are raised.

6 CONCLUSION

In this work, we presented a holistic method to guide domain experts in configuring IoT environments and applications. The method is composed of a toolbox and a business process-based approach using BPMN 2.0. The presented toolbox is the foundation of our method and contains the common building blocks of the IoT, which in our approach we named BBs BBIs. We also provide a formalization of BBs, BBIs, and Requirements. Afterward, we present our holistic method describing it step-by-step.

We developed a web-based prototype to evaluate or toolbox and method. The chosen architecture and technologies aim for a modular, scalable, and extensible prototype to be an open-source platform in the future. Although being technology independent, the motivation of implementing the prototype with extensible technologies was achieved and detailed in Chapter 4, but in future work, the used technologies might be reviewed. The developed prototype uses modern technologies to implement the formalized components of this approach, such as BBs, BBIs, and Requirements and integrates with a well-known BPMS process engine for its execution through a Microservice Architecture and programming languages such as JavaScript and Python.

As observed during the evaluation of the approach, the proposed method can provide many insights on which IoT sensors, actuators, gateways, and other devices, besides communication protocols, IoT platforms, and so on, can be selected to compose a specific smart environment. Thus, our approach intends to reduce the complexity and development time of building IoT environments. However, the time-reducing measure would be possible only by evaluating a variety of IoT environments, which is expected in future works.

We evaluate our approach by conducting it step-by-step to a real-world case study. The application of our method is intended to configure an IoT environment of an indoor localization system for a Smart Factory. Along with the applicability of our approach, we evaluate the prototype going through the method. In the final, with the experiment, we could answer the research questions of this works.

This work becomes relevant when realizing an abstract form to concept the massive variety of IoT devices and technologies into BBs and BBIs, helping domain experts to find suitable components for desired smart environments. Moreover, this work addresses the mutual benefit of the intersection of the areas of IoT and BPM regarding the configuration of smart environments.

As a continuation of this work, we aim to extend many real-world case studies to evaluate our approach pragmatically. Besides, enhance the prototype by increasing the number of pre-defined BBs, BBIs, and Requirements, develop a robust recommender system, integrate different BPMS and turn the prototype BPMS-agnostic, and finally turn it into an open-source platform.

In conclusion, as it stands, the presented holistic method has shown promising to cope with the complexity of the IoT domain and help domain experts through the whole configuration process of IoT environments. Furthermore, besides the mentioned points of improvements, this study can be used as a ground for future researches of IoT, BPM, Software Engineering, Conceptual Modeling, and Recommender Systems fields. As such, the value of this work is not restricted to the scope of this study but instead encourages the emergence of complementary researches.

The main contribution of this work is a holistic method to help domain experts through the process of configuring IoT environments and applications with the guidance of a toolbox and a business process-based approach. Besides, web-based prototype was developed.

Finally, part of this work was presented on the 39th International Conference on Conceptual Modeling ER Forum, Demo and Posters 2020 (FRIGO et al., 2020).

REFERENCES

AGGARWAL, S. Modern web-development using reactjs. **International Journal of Recent Research Aspects**, v. 5, n. 1, p. 133–137, mar. 2018.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A systematic mapping study in microservice architecture. In: **2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)**. [S.l.: s.n.], 2016. p. 44–51.

ANTONIOU, G.; HARMELEN, F. van. Web Ontology Language: OWL. In: ____. **Handbook on Ontologies**. [S.l.]: Springer Berlin Heidelberg, 2004. p. 67–92.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, p. 2787–2805, 10 2010.

AUGUSTIN, A. et al. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. **Sensors**, v. 16, p. 1466, 10 2016.

BALEN, J.; ŽAGAR, D.; MARTINOVIC, G. Quality of service in wireless sensor networks: A survey and related patents. **Recent Patents on Computer Science**, v. 4, p. 188–202, 09 2011.

BAUER, M. et al. Internet of things – architecture iot-a deliverable d1.5 – final architectural reference model for the iot v3.0. In: . [S.l.: s.n.], 2013.

BERMUDEZ-EDO, M. et al. IoT-Lite: a lightweight semantic model for the internet of things and its use with dynamic semantics. **Personal and Ubiquitous Computing**, v. 21, n. 3, p. 475–487, jun. 2017.

BHADDURGATTE, R. C.; B.P, V. K. A Review: QoS Architecture and Implementations in IoT Environment. **Research & Reviews: Journal of Engineering and Technology**, v. 2016, p. 6–12, 2016.

Bormann, C.; Castellani, A. P.; Shelby, Z. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. **IEEE Internet Computing**, v. 16, n. 2, p. 62–67, 2012. ISSN 1941-0131.

BOSCHERT, S.; ROSEN, R. Digital twin—the simulation aspect. In: ____. **Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers**. Cham: Springer International Publishing, 2016. p. 59–74. ISBN 978-3-319-32156-1. Disponível em: <https://doi.org/10.1007/978-3-319-32156-1_5>.

BULIK, M. A. **FoFdation- Foundation for Smart Factory of the Future**. 2017. [Online; accessed 9. Oct. 2021]. Disponível em: <https://wayback.archive-it.org/12090/20170401085818/https://ec.europa.eu/digital-single-market/en/blog/fofdation-foundation-smart-factory-future>.

CAMUNDA. **Camunda**. 2021. <https://camunda.com/>.

Camunda Platform community. **Introduction | docs.camunda.org**. 2021. [Online; accessed 26. Oct. 2021]. Disponível em: <https://docs.camunda.org/manual/7.12/introduction>.

CHOURABI, H. et al. Understanding Smart Cities: An Integrative Framework. In: **2012 45th Hawaii International Conference on System Sciences**. [S.l.: s.n.], 2012. p. 2289–2297.

COOK, D. J.; DAS, S. K. How smart are our environments? an updated look at the state of the art. **Pervasive and Mobile Computing**, v. 3, n. 2, p. 53–73, 2007. ISSN 1574-1192. Design and Use of Smart Environments. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1574119206000642>.

COSTA, B.; PIRES, P. F.; DELICATO, F. C. Specifying Functional Requirements and QoS Parameters for IoT Systems. In: **2017 IEEE DASC/PiCom/DataCom/CyberSciTech**. [S.l.: s.n.], 2017. p. 407–414.

COX, C. **An Introduction to LTE: LTE, LTE-Advanced, SAE, VoLTE and 4G Mobile Communications**. Wiley Telecom, 2014. Disponível em: <https://ieeexplore.ieee.org/book/8039851>.

Docker. **Empowering App Development for Developers**. 2021. <https://www.docker.com>. [Online; accessed 17. Sep. 2021].

DOMINGOS, D.; MARTINS, F. Using bpmn to model internet of things behavior within business process. **International Journal of Project Management**, v. 5, p. 39–51, 01 2017.

DUMAS, M. et al. **Fundamentals of Business Process Management, Second Edition**. [S.l.]: Springer, 2018.

FEDOSEJEV, A. **ReactJS by Example - Building Modern Web Applications with React**. [S.l.]: Packt Publishing, 2015.

FRANCO DA SILVA, A. C. et al. Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: **Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER)**. [S.l.]: SciTePress Digital Library, 2017. p. 358–367.

FRANCO DA SILVA, A. C.; HIRMER, P. Models for Internet of Things Environments—A Survey. **Information**, v. 11, n. 10, 2020.

FRIGO, M. et al. A Toolbox for the Internet of Things–Easing the Setup of IoT Applications. In: **Proceedings of the ER Forum, Demo and Posters 2020 co-located with 39th International Conference on Conceptual Modeling**. [S.l.: s.n.], 2020.

GOMEZ, C. et al. Internet of Things for enabling smart environments: A technology-centric perspective. **Journal of Ambient Intelligence and Smart Environments**, v. 11, p. 23–43, 01 2019.

Google. **gRPC API reference**. 2021. <https://grpc.io/docs/>.

HALLER, S.; KARNOUSKOS, S.; SCHROTH, C. The internet of things in an enterprise context. In: . [S.l.: s.n.], 2008.)5468, p. 14–28. ISBN 978-3-642-00984-6.

HARPER, R. **Inside the Smart Home**. [S.l.]: Springer Science & Business Media, 2006.

HIRMER, P. et al. Automating the Provisioning and Configuration of Devices in the Internet of Things. **Complex Systems Informatics and Modeling Quarterly**, Online, v. 9, p. 28–43, 2016. ISSN 2255 - 9922.

HOVLAND, P. et al. A quality of service approach for high-performance numerical components. In: . [S.l.: s.n.], 2003.

Hunkeler, U.; Truong, H. L.; Stanford-Clark, A. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. In: **2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)**. [S.l.: s.n.], 2008. p. 791–798.

IEEE. Ieee standard for an architectural framework for the internet of things (iot). **IEEE Std 2413-2019**, p. 1–269, 2020.

ISINKAYE, F.; FOLAJIMI, Y.; OJOKOH, B. Recommendation systems: Principles, methods and evaluation. **Egyptian Informatics Journal**, v. 16, n. 3, p. 261–273, 2015. ISSN 1110-8665. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1110866515000341>.

JANIESCH, C. et al. The Internet-of-Things Meets Business Process Management: Mutual Benefits and Challenges. 09 2017.

JARAMILLO, D.; NGUYEN, D.; SMART, R. Leveraging microservices architecture by using Docker technology. **ResearchGate**, p. 1–5, Mar 2016.

JEDERMANN, R.; LANG, W. The benefits of embedded intelligence – tasks and applications for ubiquitous computing in logistics. In: . [S.l.: s.n.], 2008. p. 105–122. ISBN 978-3-540-78730-3.

JUSAS, N. **Feature model-based development of Internet of Things applications**. 2017. [Online; accessed 22. Aug. 2021]. Disponível em: <https://epubl.ktu.edu/object/elaba:23031986>.

KHALED, A. E. et al. IoT-DDL–Device Description Language for the "T" in IoT. **IEEE Access**, v. 6, p. 24048–24063, 2018. ISSN 2169-3536.

LIGHT, R. A. Mosquitto: server and client implementation of the mqtt protocol. **Journal of Open Source Software**, The Open Journal, v. 2, n. 13, p. 265, 2017. Disponível em: <https://doi.org/10.21105/joss.00265>.

MAYER, S. et al. Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning. In: **2014 International Conference on the Internet of Things (IOT)**. [S.l.]: IEEE, 2014. p. 6–8. ISBN 978-1-4799-5154-3.

MCEWEN, A.; CASSIMALLY, H. **Designing the Internet of Things**. [S.l.]: Wiley Publishing, 2013. ISBN 111843062X, 9781118430620.

MILLMAN, K.; AIVAZIS, M. Python for Scientists and Engineers. **Comput. Sci. Eng.**, v. 13, p. 9–12, May 2011.

MongoDB Inc. **MongoDB API reference**. 2021. <https://docs.mongodb.com/manual/>.

MOZILLA. **JavaScript**. 2015. [Online; accessed 2. Oct. 2021]. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.

NEWMAN, S. **Building Microservices**. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015. ISBN 978-1-49195035-7. Disponível em: <https://www.oreilly.com/library/view/building-microservices/9781491950340>.

OASIS. **Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0**. [S.l.], 2013.

OGC. **Sensor Model Language (SensorML)**. 2008. <http://www.opengeospatial.org/standards/sensorml>.

OLIPHANT, T. E. Python for Scientific Computing. **Comput. Sci. Eng.**, IEEE, v. 9, n. 3, p. 10–20, Jun 2007. ISSN 1558-366X.

OMG. **About the Business Process Model and Notation Specification Version 2.0.2**. 2014. <https://www.omg.org/spec/BPMN/2.0.2/About-BPMN>. [Online; accessed 14. Aug. 2021].

OpenJS Foundation. **Node.js API reference**. 2021. <https://nodejs.org/en/about/>.

PAVLENKO, A. et al. Micro-frontends: application of microservices to web front-ends. 05 2020.

RABBITMQ. **Rabbit Message Queuing (RabbitMQ)**. 2007. <https://www.rabbitmq.com/documentation.html>.

RADEMACHER, F.; SACHWEH, S.; ZüNDORF, A. Differences between model-driven development of service-oriented and microservice architecture. In: **2017 IEEE International Conference on Software Architecture Workshops (ICSAW)**. [S.l.: s.n.], 2017. p. 38–45.

RAJ, A.; JASMINE, K. Building Microservices with Docker Compose. **International journal of analytical and experimental modal analysis**, XIII, p. 1215, May 2021.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Pearson, 2016. (Always learning). ISBN 9781292153964. Disponível em: <https://books.google.com.br/books?id=XS9CjwEACAAJ>.

SCHöNIG, S. et al. IoT meets BPM: A Bi-directional Communication Architecture for IoT-Aware Process Execution. **Software and Systems Modeling**, 02 2020.

SHROUF, F.; ORDIERES, J.; G.MIRAGLIOTTA. Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm. In: **2014 IEEE International Conference on Industrial Engineering and Engineering Management**. [S.l.: s.n.], 2014. p. 697–701.

SOMMERVILLE, I. **Software Engineering, 10th Edition**. London, England, UK: Pearson, 2015. ISBN 978-0-13758669-1. Disponível em: <https://www.oreilly.com/library/view/software-engineering-10th/9780137586691>.

SURI, K. et al. Semantic Framework for Internet of Things-Aware Business Process Development. In: **2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)**. [S.l.: s.n.], 2017. p. 214–219.

TAIVALSAARI, A.; MIKKONEN, T. A Roadmap to the Programmable World: Software Challenges in the IoT Era. **IEEE Software**, v. 34, n. 1, p. 72–80, 2017.

TEKINERDOGAN, B.; KOKSAL, O. Pattern based integration of internet of things systems. In: ____. [S.l.: s.n.], 2018. p. 19–33. ISBN 978-3-319-94369-5.

TORRES, V. et al. Modeling of IoT devices in Business Processes: A Systematic Mapping Study. In: . [S.l.: s.n.], 2020. p. 221–230.

TOWNSEND, K. et al. **Getting Started with Bluetooth Low Energy**. Sebastopol, CA, USA: O'Reilly Media, Inc., 2014. ISBN 978-1-49194951-1. Disponível em: <https://www.oreilly.com/library/view/getting-started-with/9781491900550>.

UFRN. **Controle na palma da mão**. 2021. [Online; accessed 27. Oct. 2021]. Disponível em: <https://ufrn.br/imprensa/materias-especiais/47271/controle-na-palma-da-mao>.

ULLAH, S. et al. Applications of uwb technology. 11 2009.

VENčKAUSKAS, A. et al. Modelling of internet of things units for estimating security-energy-performance relationships for quality of service and environment awareness: Modelling of iot units for estimating quality of service. **Security and Communication Networks**, v. 9, 06 2016.

VERMESAN, O.; FRIESS, P. **Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems**. [S.l.]: River Publishers, 2013.

VIPUL, M.; SONPATKI, P. **ReactJS by Example - Building Modern Web Applications with React**. [S.l.]: Packt Publishing, 2016.

WEERAWARANA, S. et al. **Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More**. USA: Prentice Hall PTR, 2005. ISBN 0131488740.

YELAMARTHI, K.; AMAN, M. S.; ABDELGAWAD, A. An Application-Driven Modular IoT Architecture. **Wireless Commun. Mobile Comput.**, v. 2017, n. 1, p. 1–16, Jan 2017.

ZANELLA, A. et al. Internet of things for smart cities. **IEEE Internet of Things Journal**, v. 1, n. 1, p. 22–32, 2014.