BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

SUPPORT FOR USER-DEFINED FUNCTIONS IN A VISUAL PROGRAMMING LANGUAGE

PODPORA UŽIVATELSKY DEFINOVANÝCH FUNKCÍ VE VISUÁLNÍM PROGRAMOVACÍM JAZYCE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **IVAN CHODÁK**
AUTOR PRÁCE

SUPERVISOR **Ing. PETR JOHN**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

161063

Institut:	Department of Information Systems (DIFS)
Student:	**Chodák Ivan**
Programme:	Information Technology
Title:	**Support for User-Defined Functions in a Visual Programming Language**
Category:	Information Systems
Academic year:	2024/25

Assignment:

1. Study the Internet of Things (IoT) and Smart City.
2. Study the principles of visual programming language (VPL) representation, focusing on visual languages applicable to IoT.
3. Analyze the requirements for web components that allow the definition of custom end-user defined functions in the Pocketix VPL editor in a user-friendly way.
4. Design an appropriate way to work with user-defined functions, including evaluation and visualizations.
5. Implement and integrate the proposed solution within the Pocketix project.
6. Test the resulting solution for usability. Suggest possible extensions.

Literature:

- Greengard, S. (2015). The Internet of Things. MIT Press, ISBN 978-026-2527-736.
- Badii, C., Bellini, P., Difino, A., Nesi, P., Pantaleo, G., & Paolucci, M. (2019). Microservices suite for smart city applications. *Sensors*, *19*(21), 4798.
- Hynek, J. a spol. (2023). Služby pro systém řízení a monitoringu vody v retenčních nádržích. Research report. Brno University of Technology.
- Kuhail, M. A., Farooq, S., Hammad, R., & Bahja, M. (2021). Characterizing visual programming approaches for end-user developers: A systematic review. IEEE Access, 9, (pp. 14181-14202).
- Ray, P. P. (2017). A survey on visual programming languages in internet of things. *Scientific Programming*, *2017*.
- John, P. (2024). Optimising processes in IoT. Brno. PhD thesis proposal University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Tomáš Hruška, CSc.
- Bureš, M. (2024). Systém pro zpracování dat z chytrých zařízení, Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Hynek, Ph.D.

Requirements for the semestral defence:

- Points 1 - 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**John Petr, Ing.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	22.10.2024

Abstract

This thesis explores the design and implementation of user-defined functions (UDFs) in a visual programming language (VPL) within the context of the Internet of Things (IoT) and Smart Cities. It is built on top of an existing Pocketix vpl-editor tool and integrated into the RIoT system. The primary objective is to enable non-technical users to create, manage, and execute custom procedures, allowing for control of various IoT devices in smart environments intuitively. By analysing the implementation against existing frameworks like Blockly and Node-RED, the thesis identifies key requirements for accessibility, flexibility, and seamless user interaction to accommodate them. The proposed solution is integrated into the RIoT system, providing a user-friendly way for creating reusable workflows through a visual editor. The resulting framework promotes logical encapsulation, scalability, and efficient interaction via different IoT devices in the environment, empowering users to define reusable procedures with ease. These user procedures can be executed via the RIoT interpret into a Go code, suitable for IoT environments.

Abstrakt

Táto práca sa zaoberá návrhom a implementáciou užívateľsky definovaných funkcií (UDF) vo vizuálnom programovacom jazyku (VPL) v kontexte Internetu vecí (IoT) a inteligentných miest. Stavia na existujúcom Pocketix vizuálnom editore a je integrovaná do RIoT systému. Hlavným cieľom je umožniť technicky nezdatným užívateľom intuitívne vytvárať, spravovať a využívať vlastné procedúry spravujúce a ovládajúce IoT zariadenia v múdrych ekosystémoch. Analýzou existujúcej implementácie v kontraste voči existujúcim systémom, ako sú Blockly a Node-RED, identifikuje kľúčové požiadavky na prístupnosť, flexibilitu a plynulú interakciu pre užívateľa. Navrhované riešenie je integrované do systému RIoT a poskytuje užívateľsky prívetivý spôsob pre na vytváranie opakovane použiteľných procedúr prostredníctvom vizuálneho editoru. Výsledné riešenie podporuje logickú enkapsuláciu, škálovateľnosť a efektívnu interakciu so zariadeniami IoT, čím umožňuje používateľom jednoducho definovať znova použitelné procedúry. Tieto užívateľské procedúry môžu byť spustené za pomoci RIoT interpretu v Go kóde, ktorý je vhodný pre prácu v múdrych prostrediach.

Keywords

visual programming languages, IoT, Smart Cities, end–user defined functions

Klíčová slova

vizuálne programovacie jazyky, IoT, inteligentné mestá, užívateľsky definované funkcie

Reference

CHODÁK, Ivan. *Support for User-Defined Functions in a Visual Programming Language*. Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr John

Rozšířený abstrakt

V súčasnosti prechádza svet zrýchlujúcou sa digitalizáciou, v rámci ktorej zohráva Internet vecí (IoT) kľúčovú úlohu pri transformácii tradičných prostredí na inteligentné systémy schopné autonómneho rozhodovania a fungovania. Vývoj takýchto systémov si však často vyžaduje odborné znalosti z oblasti programovania, čo výrazne obmedzuje zapojenie bežných používateľov a spomaľuje adopciu inteligentných riešení v domácnostiach, školách, firmách a samosprávach. Napriek množstvu dostupných platforiem je stále cítiť absenciu nástroja, ktorý by umožnil jednoduchú a zrozumiteľnú tvorbu automatizačnej logiky bez nutnosti písania kódu zastrešujúcu rôzne zariadenia pod jedným systémom. Táto bakalárska práca skúma potreby užívateľa a existujúcich riešení za cieľom vytvorenia návrhu a je jeho implementácie pre rozšírenie podpory vytvárania, spravovania a parametrizávie používateľsky definovaných funkcií (User-Defined Functions – UDF) v rámci v organizácie Pocketix.

Základným cieľom tejto práce je uľahčiť definovanie vlastnej logiky správania IoT zariadení prostredníctvom intuitívneho grafického rozhrania, ktoré sprístupní tvorbu automatizačných scenárov aj netechnickým používateľom. To je obzvlášť dôležité v kontexte narastajúceho počtu inteligentných zariadení, ktoré generujú veľké množstvo dát a zároveň poskytujú možnosti interakcie prostredníctvom aktuátorov. Úspešné nasadenie týchto zariadení však často zlyháva práve na nedostatočnej podpore používateľskej konfigurácie zariadeniami, užívateľsky prívetivým spôsobom.

Z pohľadu dostupných riešení je problém ešte zreteľnejší. Komerčné platformy síce poskytujú jednoduché rozhrania na tvorbu logiky, napríklad v štýle podmienok a akcií závyslích na nich, ale tie sú väčšinou obmedzené na úzke množiny preddefinovaných scenárov a nedovoľujú výraznejšiu modifikáciu alebo znovupoužiteľnosť vytvorených funkcií. Navyše bývajú zviazané s konkrétnymi službami alebo produktmi, čím vzniká fragmentácia a strata interoperability. Práca preto prináša open-source alternatívu umožňujúcu širšiu personalizáciu, abstrahovanie logiky a opätovnú použiteľnosť v rámci systému RIoT.

Úvod práce je venovaný prehľadu technológií IoT a ich architektonickým modelom, pričom sú zdôraznené výhody vrstveného prístupu a použitie protokolov ako MQTT či CoAP pre spoľahlivú a škálovateľnú komunikáciu. Ďalšia časť sa zaoberá vizuálnym programovaním ako prostriedkom na demokratizáciu vývoja, historickým vývojom VPL, jeho výhodami a rozdielmi oproti textovému programovaniu, ako aj analýzou konkrétnych existujúcich riešení Blockly, Node-RED a Scratch.

V centre práce stojí analýza stavu editora `vpl-for-things`, ktorý bol navrhnutý ako jednoduchý VPL editor, primárne pre IoT prostredie systému RIoT. Editor umožňoval definíciu základných procedúr, ale neumožňoval ich parametrizáciu, abstrahovanie vstupov, ani opätovné použitie mimo aktuálny kontext. Práca preto navrhuje nový prístup založený na tzv. „Skeletonize“ móde. Tento režim umožňuje označiť bloky v rámci existujúceho programu a transformovať ich do samostatnej procedúry, ktorá je automaticky parametrizovaná. Parametre sú reprezentované blokmi zariadení, ktoré sú pri použití procedúry nahradené aktuálnymi komponentmi systému RIoT. Takto je možné dosiahnuť vysokú mieru flexibility, znovupoužiteľnosti a abstrakcie.

Realizácia riešenia zahŕňa zemny vyžadujúce zásahy do viacerých úrovní systému. Frontend rozhranie bolo upravené tak, aby podporovalo výber časti programu, jeho konverziu na UDF a jeho neskoršie vloženie do iných programov. Boli implementované nové modálne okná, používateľské interakcie a systém validácie parametrov. Na úrovni backendu bol zavedený vzťah medzi entitami programov a procedúr, doplnená logika dynamického načítania iba tých procedúr, ktoré sú relevantné pre aktuálny program, čím sa znižuje pamäťová

a výpočtová záťaž. Implementácia bola integrovaná do API a databázy systému RIoT. Implementovaný návrh bol následne otestovaný rôznymi užívateľskými skupinami.

Používateľské testovanie zahŕňalo demo predstavenie systému RIoT a editoru pre Pocketix. Následne mali za úlohu vytvoriť jednoduchý program, definovať a použiť UDF. Prebehli tri testovania, každé s iným zameraním s ohľadom na testovaciu skupinu. Prvý test zahŕňal technicky zdatného užívateľa ktorý mal splniť úlohu bez bližšej inštruktáže s cieľom zistiť intuitívnosť aplikácie. Následne bolo prevedené testovanie s cieľovým užívateľom kde boli hodnotené prívetivosť a prínosnosť. Posledný test zahŕňal študemtov technickej unverzity kde sa zbierali názory a nápady na potenciálne ďalšie rošírenia a zlepšenia. Testy potvrdili, že zavedenie Skeletonize režimu znižuje počet potrebných interakcií, zrýchľuje vývoj a zároveň zvyšuje užívateľský komfort. Parametrizácia umožnila jednoduché znovupoužitie definovaných procedúr v iných kontextoch, čo výrazne podporuje modularitu.

Výsledky jednoznačne ukazujú, že navrhnuté rozšírenie implementované v tejto práci spĺňa stanovené ciele a zároveň otvára nové možnosti pre budúci rozvoj systému. Práca prináša funkčnú implementáciu riešenia a navrhuje budúce rozšírenia a zlepšenia na základe testovania. Medzi ne patrí zavedenie podpory pre vytvorenie centrálneho úložiska UDF na ulahčenie výberu a zdieľanie procedúr medzi používateľmi, ako aj vylepšené UI a optimalizácia rozhrania pre mobilné zariadenia.

Support for User-Defined Functions in a Visual Programming Language

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. John Petr. I have listed all the literary sources, publications and other sources which were used during the preparation of this thesis.

.....
Ivan Chodák
May 21, 2025

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Ing. John Petr, for his exceptional guidance, support, and encouragement throughout the development of this bachelor's thesis. His expertise, insightful feedback, and tireless availability were invaluable in shaping delivering this work. His profound patience and thorough explanations, paired with his commitment to high-quality and well-executed ideas, have been truly inspiring. I am deeply grateful for the opportunity to write my thesis under his supervision and the experiences gained along the way.

I would also like to extend my heartfelt thanks to my mother for her unwavering support and encouragement. Her belief in my abilities and constant motivation gave me the strength to overcome challenges and stay focused on my goals.

Without their support, this thesis would not have been possible. Thank you.

Contents

List of Figures

Chapter 1

Introduction

The ever-growing number and range of interconnected devices interacting with the real world opens up doors to many possibilities for quality-of-life improvements, starting from individual users, families or corporations all the way up to cities and countries. These devices appear in our everyday lives in various forms and shapes. Some more obvious such as smart watches monitoring our heartbeat, steps, saturation and other real-time data. Other devices help us live more comfortable lives such as thermostats in our homes. They monitor the temperature outside and inside the building, utilizing the data from their inputs to adjust the heating according to the need. Many are hidden both in plain sight and out of it in the form of more specialized systems like industrial sensors, environmental monitoring tools, and security systems. Each of these devices plays a pivotal role in capturing, processing, and transmitting data, enabling the creation of smart environments.

These smart environments are created by a network of connected cooperating devices referred to as The Internet of Things (IoT). The Internet of Things refers to the broader concept of interconnected devices communicating over the Internet, providing capabilities for automatization, streamlining, and optimization of tasks, thus improving their overall efficiency. The IoT extends over most major domains such as agriculture, healthcare, transportation, smart homes or a niche application custom tailored by the user. Through seamless device-to-device communication, IoT creates opportunities for smarter, more connected systems capable of functioning with minimal human intervention. However, to minimize human intervention, we need to provide the user with a scalable and long-lasting solution. A solution that can facilitate and manage different types of sensors and IoT devices while letting the user define their interactions or custom routines through end-user-defined functions.

One of its significant applications in the near future is in the creation and management of Smart Cities. Through the various integration of different IoT-integrated devices and sensors, cities can become more efficient, sustainable, and responsive to the needs of residents or a random event that can develop in moments. A few of the many examples include intelligent traffic management, allowing for a better flow or prioritization of the public transport lines, automated waste collection and street cleaning, and enhanced public safety or fire prevention systems. Such use cases demonstrate the transformative potential of IoT, especially when deployed on a large scale in urban areas.

As the IoT ecosystems grow more complex and more users are exposed to them and their workings, so does the need for the aforementioned end-user-defined functions. This need stems from the aim to provide the user with greater and better control over the behaviour and interactions of IoT devices. However, to accommodate this functionality

and to provide this control, a user-friendly interface that allows for a swift and comfortable definition of custom functions to the end-user without deep technical knowledge or a steep learning curve is needed. This must be coupled with a uniform control mechanism that works across different devices, systems, and sensors, which is crucial to ensuring that users can fully leverage the potential of IoT. It must be developed in such a way as to be able to serve different types of users in various roles, integrating all sorts of inputs and outputs in a simple, meaningful manner.

This brings us to the motivation for this work. The need for support for User-Defined-Functions (UDFS) in a Visual Programming language (VPL), namely in the Pocketix project, which provides a block-based visual editor for the Pocketix VPL. The VPL provides a simplified, graphical way for users to interact with smart systems, reducing the complexity of coding while making customisation accessible to non-technical users. By integrating UDFS into a VPL, we enable users to more easily define their own logic, empowering them to control and customise the behaviour of devices within a smart environment. This thesis is built on top of a Pocketix VPL Editor project for which a front-end support for reusable UDFs and a back-end integration into the Real-time IoT (RIoT) data processing information system will be implemented.

The following chapters of this thesis will provide insight into the IoT and its potential usage in the creation and implementation of Smart Cities, explore the current and overall challenges of fully utilizing the customization of IoT systems, analyze the existing VPL options and their UDFs capabilities in regard to IoT integration, propose and develop a way for the user to create, store, and reference UDFs via VPL Editor in the Pocketix RIoT system. This will include both the front-end presentation of the EUDFs as well as their back-end declaration and integration within the existing Pocketix and RIoT structure.

Firstly, Chapter 2 aims to provide an overview of the IoT landscape and its role in Smart City and its development. This chapter outlines the fundamental aspects of IoT, illustrating some of the architectures and technologies commonly used and how IoT devices interact to enhance automation and efficiency, mainly examining the use in Smart City environments. It discusses key infrastructure improvements like traffic optimisation and energy management, showcasing how IoT technologies contribute to smarter urban systems and a higher quality of life.

Secondly, Chapter 3 examines what VPLs are, their history and evolution, and the existing VPL solutions out there. The main focus will be on how they tackle the support for EUDFS and the different advantages and disadvantages arising from the given solution. This chapter will also discuss the design principles and usability considerations that are critical for enabling non-technical users to effectively interact with IoT systems to their fullest potential.

Within Chapter 4, we dive into the analysis of the current Pocketix VPL Editor implementation, what it offers in terms of EUDFs support, and what key requirements need to be fulfilled to elevate the functionality provided, while adhering to the original idea. Different aspects of existing VPLs and their capabilities for integration and creation of IoT applications will be evaluated, drawing on the conclusions and findings from the 3, focusing mainly on their ease of defining custom functions in smart environments. Existing VPLS will be analysed on how they can be better optimised to handle IoT's growing demands for user-defined automation while comparing it with the current state of Pocketix and how these findings translate to possible future add-ons.

Chapter 5 proposes an extension for the Pocketix allowing for user-defined functions and its integration into the RIoT system. Based on prior analysis, we will propose and

implement a user-friendly VPL design, rooted in and built on top of the current Pocketix VPL Editor. Thus, offering users an intuitive interface to visually define device interactions and routines in the RIoT system. The solution emphasises reusability while enabling more control and automation across smart environments utilising IoT devices.

Followed by Chapter 6 where the solution's technical aspect is explained in more detail. The implementation goes over the front-end and back-end solutions for both the VPL Editor and its integration into RIoT.

In Chapter 7, the goals of the thesis are re-examined to set the testing parameters. A test case is proposed and tested with the relevant target audience, with each test being briefly described. Findings are evaluated and used for proposing future add-ons to either modify the existing implementation or extend it with new elements and features the users would appreciate. The last Chapter 8 will go over the overall evaluation of the goals set in this thesis and the level to which they were met.

Chapter 2

IoT and Smart Cities

The phrase "Internet of Things" started its prosperous life in the year 1999 as the title of a presentation by Kevin Ashton at Proctor & Gamble (P&G) [3]. In it, he proposed linking the new RFIDs in P&G's supply chain to the internet. This was more than enough to fulfil his goal to attract executive attention. He also gave an important insight that has been misunderstood in the years to come and even today. In his eyes, the Internet of Things was not just a simple barcode upgrade or a way to speed up the toll roads. He envisioned it as a means to embrace the potential of computers, empowering them to observe, identify, and eventually understand the world with little to no human interaction [3].

The Internet of Things represents a paradigm [4] where physical objects are interconnected with virtual ones through various communication protocols and technologies to provide more advanced services utilizing data collection, processing and evaluation, and distribution.

Another similar IoT definition is as an ecosystem [31] of devices communicating seamlessly through implementing various technologies, enabling real-time analytics and decision-making in diverse applications and environments.

The Internet of Things as a concept has experienced massive growth and transformation in how digital systems and real objects interact, paving the way for advanced connectivity and allowing for better and more high-end automation [4].

The majority of sources come to the consensus that the main devices enabling these advancements depend on integrating several technological and communication solutions. The most relevant devices and Iot objects taking part in this growth process are identification and tracking devices, various sensors depending on the desired utility, actuator networks providing the backbone for the intercommunication of devices, and enhanced communication and security protocols to protect the user against unauthorised access.

The synergy of all the information obtained from various sensors and reading devices, processed by the correctly chosen technology for the given task, and the ability to act on the output in an appropriate and timely manner offers us a unique possibility to transform and build the world around us in a manner never seen before. The right tool must be chosen for the task at hand after considering the needs and limitations of the project at hand. Facilitating this opportunity for quality of life improvement to as many people as possible is important to me. Therefore, this chapter will examine the technologies, applications, and examples within IoT with this goal in mind, focusing on those I find most relevant to explain the concept without overwhelming the reader. These are not all the technologies used in IoT and smart environments, but they represent the foundational pillars from which a well-tailored solution can be crafted.

2.1 Architectures

As IoT is present across many fields, each of which requires varying levels of complexity, the need for adequate architecture fitting the given situation arises. There are numerous publications talking about the different architectures [49, 15, 31] with layer-based division.

This layer logic allows for encapsulation and separation, providing us with varying levels of abstraction and functionality tailored to meet various needs in more complex systems or user-defined implementations of IoT solutions. I will examine three layer-based architectures to give a cross-section overview across the abstraction level, which can be found in the smart environments, starting with a simple three-layer architecture to explore the basic concepts, expanding on with the five-layer architecture, which provides two more dedicated layers allowing for further specification and optimisation. To top it off, the seven-layer architecture will be examined, exploring technologies and concepts best utilised in large-scale IoT environments such as Smart Cities.

Three-layer Architecture

The simplest and thus also one of the first representations is the three-layer architecture seen on the left side in Figure 2.1, made up of the following layers [4]. The perception, the transport, and the application layer. Each serves a separate purpose to accommodate the IoT implementation of collecting data, evaluating it, and acting on it in an environment. This structure allows for an easy abstraction over small and simple IoT systems or applications where just the basic data collection, transmission, and delivery to end-user applications is sufficient [49]. While being lightweight and compact, it suffers from a lack of scalability and data processing capabilities. I will now examine the layers more closely, observing their part, principles,and purpose.

The perception layer, also referred to as the Device layer, is the foundation block of the IoT architecture, allowing the device to sense various aspects of the physical environment and data collection from it via capturing and monitoring devices [17]. Devices such as Sensors, actuators, RFID tags, and other IoT enable devices tasked with obtaining data by converting the physical information into digital data and signals, allowing for it to be propagated, processed, and acted upon in the higher layers. These play a role in contributing to the creation of the main advantage of the perception layer, the ability to connect a wide range of devices to the IoT network, thus enabling real-time data acquisition and monitoring [24]. Despite this, it faces a number of challenges in various areas with the need to find balance in them. A major challenge to this layer is its security as the devices are suspectable to physical and cyber attacks [24], due to the aim of creating them in such a manner as to be low-cost and energy-efficient. This is also reflected in the often low computational resources and power constraints.

The transport layer, also referred to as the network or communication layer, is responsible for the transmission of the collected data from the perception layer for further processing and subsequent storage [4]. It serves as the medium for data flow within the IoT system, using different communication protocols such as Wi-Fi, Bluetooth, Zigbee, or 5G to facilitate all the necessary connectivity and communication [49]. The combination of various technologies and protocols allows the network layer to provide data transfer across both short-range and long-range distances [4]. The most important aspects of the transport layer are definitely the scalability and adaptability, serving as its main advantage, and the potential data integrity, latency or overall connectivity issues, and security [19] concerns

serving as its disadvantage. These issues become more apparent, especially in managing larger IoT operations consisting of more subsystems, each relying on a multitude of various devices and sensors as the data can be intercepted, manipulated, or corrupted by malicious actors.

The application layer is the topmost layer of this IoT architecture. Its purpose is to deliver the final result of processed data to the end-user application [50]. It is also a layer responsible for not just the presentation but also the interpretation of given data for specific application purposes across various fields. The application layer is the key element in IoT providing real added value [1]. That is achieved by turning the raw captured data into actionable and meaningful insights that the user or a subsequent automated system can act upon. While supporting a wide range of applications, each relying on and requiring different data processing steps, storage needs, as well as various custom user interface elements, faces challenges in data heterogeneity and integration [1]. Furthermore, the security and privacy concerns are highlighted in contrast to other layers as this layer directly facilitates direct interaction with data and systems.

Five-layer Architecture

To address the growing complexity and size of the IoT systems, a solution with more precise control and overview of the system was needed [49, 17]. This is where the five-layer architecture, the middle one in Figure 2.1, with two additional layers for the processing and business sandwiching the application layer, comes into play [1]. These extra layers provide enhanced data management, storage, analysis, and application-specific logic, making them more suitable for more complex operations supporting a broader range of functionalities and specific needs. Building on the three-layer architecture, the following two layers are added to boost computing capability and ease of use. The first one is the processing layer [1] and the second is the business logic layer [1]. This expansion to the three-layer architecture provides better control and organisation as it provides better encapsulation and new possible implementation solutions, thus being more scalable, adaptable, and overall offer a better solution for facilitating bigger and more robust systems. The added layers are examined in more depth to see how they affect the IoT system and expand its capabilities.

The **processing layer**, also referred to as the middleware Layer, enhances the standard IoT architecture by adding a dedicated space for data analysis, storage and decision-making [6]. This is accomplished by leveraging cloud and edge computing to better manage and process the massive data volumes generated by the IoT device. While cloud computing offers scalable storage and processing power by having the remote servers perform data handling and processing, edge computing [6] offers a similar ability of quick data processing by utilizing local computing resources at the network's edge.

This combination creates Fog computing, which serves to reduce latency as the data can be processed immediately by edge computing before being sent to the cloud for remote storage or processing. This is especially useful for tasks requiring immediate responses. In addition, it also supports advanced data processing techniques, including machine learning and various analytics, allowing the raw data to be transformed into useful insight [6]. While the processing layer offers many enhanced capabilities and benefits to IoT systems, it also presents similar challenges regarding ensuring data privacy, maintaining processing speed, and handling increased storage demands, making it essential to balance resources carefully to optimize performance in large-scale applications with its utility [17].

The **business layer** [50], accommodated by the top layer in the five-layer Iot architecture, acts as a decision-making hub. This is where the insights from lower layers get transformed into concrete business actions aligned with the strategic goals set by the user or system controls. The business layer can coordinate multiple subsystems to react to a newly created situation based on the insights from lower layers, allowing for a swift, tailor-made solution to each situation as it arises. In addition to integration, it can manage and enforce application-specific rule sets and compliances, making it easier to meet the operational and regulatory demands [1]. For instance, in an Iot smart healthcare environment, the business layer enforces strict access protocols which aim to protect patient privacy, in line with regulations like HIPAA, while at the same time optimising and streamlining necessary data logging and subsequent action taking based on them.

Seven-layer Architecture

The seven-layer architecture is created by breaking each of the network and middleware layers from a five-layer architecture into two more specified layers. These are the Connectivity and Edge Computing Layer alongside the Data Accumulation and Abstraction Layer seen in Figure 2.1. These layers enhance performance while reducing system load [49, 31].

The Edge Computing Layer [6] processes data closer to its source, reducing latency and bandwidth usage by handling relevant information locally. This offloads pressure from central servers, enabling real-time decision-making critical for applications like smart factories, where data processing is done on-site to detect defects or optimise production without cloud delays [43].

The Data Accumulation Layer [35] collects, buffers, and aggregates raw data from edge devices, ensuring consistency and preventing loss during transmission, which is vital for environments with high data volumes or intermittent connectivity.

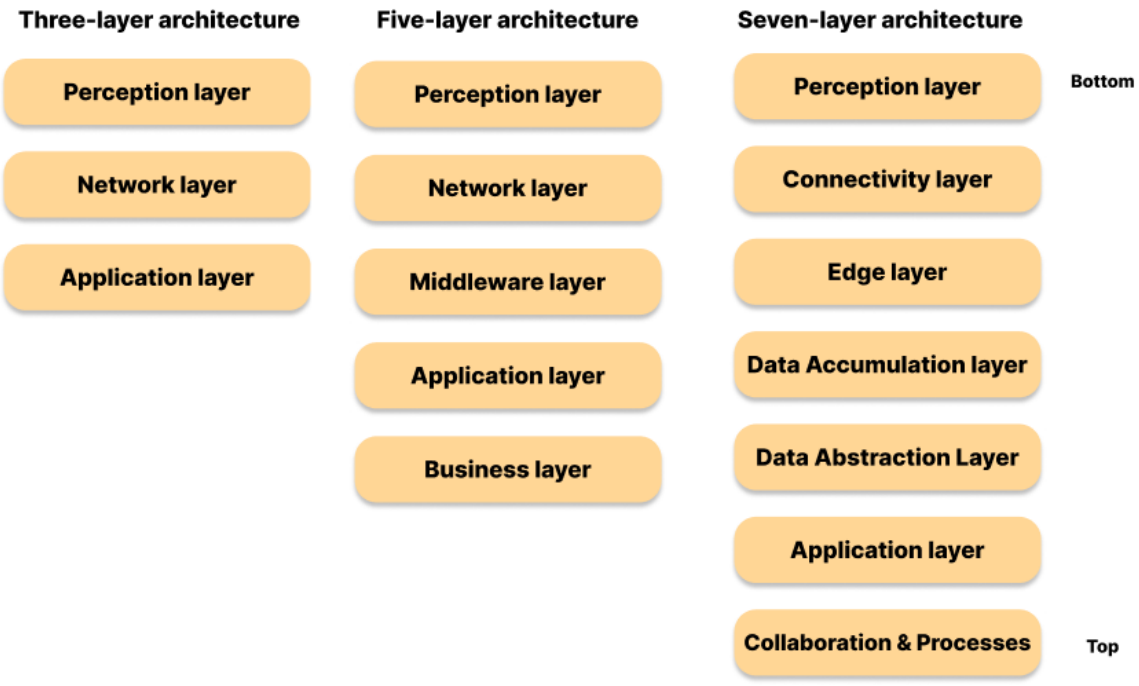

Figure 2.1: Three, Five, and Seven layer architectures

Positioned above it, the Data Abstraction Layer [17] standardises aggregated data for compatibility across diverse applications, enabling seamless integration in heterogeneous IoT systems. For instance, in Smart Cities, it unifies data from weather sensors, traffic cameras, and utility meters for urban management [26]. Together, these layers improve scalability and interoperability, addressing challenges in real-time processing and multi-source integration.

2.2 Technologies

IoT technologies needed for the implementation encompass a broad spectrum, from sensors and actuators that gather real-world data, to advanced communication protocols that enable devices to exchange information efficiently. They also include robust data processing frameworks, such as cloud and edge computing, which manage and analyze vast amounts of information. Furthermore, as IoT systems grow more complex and integrated into critical infrastructure, security and privacy technologies play an increasingly vital role in protecting data and ensuring system integrity. However, most of the applications of IoT require a tailor-made solution consisting of fitting technologies and devices.

Sensing and Data Acquisition Technologies

As mentioned in the perception layer, the ability to sense and perceive the real world is essential for the IoT to perform its designed function. This is achieved thanks to various sensing and acquisition technologies [17] interacting with and monitoring the physical environment they are located in. These technologies help us bridge the gap between the physical and digital worlds via the employment of sensors, actuators, and edge devices to monitor, quickly analyze, and respond to conditions as they arise [50].

Sensors and actuators are the main components present in the perception layer. The sensors' role is to capture real-world data and relay it further into the ecosystem. Some examples of sensing devices include but are not limited to temperature sensors,such as DS18B20 Digital Thermometer, which provide precise readings for various applications relying on the temperature data. motion, humidity, and many more parameters collected from the real world in smart environments [4]. For more environmental monitoring, humidity sensors like the DHT11 or DHT22, these specific examples provide temperature readings as well, let the user monitor the humidity of the location allowing for creation of smart farming environments or industrial applications with need of humidity overview. Light sensors, like the BH1750 Digital Ambient Light Sensor, can find a wide application from automatic blinds and light control or to control the day light cycle in agriculture. The motions sensors, like the HC-SR501 Passive Infrared (PIR) Sensor, detect changes in infrared radiation, making them ideal for security systems and smart lighting solutions. Additionally, gas sensors, like the MQ-135 Air Quality Sensor, monitor air pollution or detect hazardous gases which can be crucial in preventing accidents in industry setting or improving the life quality of the citizens in smart cities. Vibration sensors, such as the ADXL356 accelerometer, often used for early failure detection in critical infrastructure such as turbines, pumps, and conveyor belts.

Actuators, on the other hand, perform physical actions that interact with the environment based on instructions obtained after processing the raw data from sensors. These actions can include interactions such as opening or closing valves, turning switches on and off, or adjusting the thermostat settings to represent and adapt to the current situation

[2]. Some examples of the actuators used include DC gear motors, solenoid actuators, and pneumatic actuators. DC gear motors, such as the Johnson Electric DC Motor with Gearbox, are widely utilized in automated doors, conveyor systems, and robotics for delivering controlled torque and motion. Solenoid actuators, like the Kendrion LIN B Series, are frequently used in applications with need for locking, cutting, clamping, punching, positioning, diverting, holding or rotating. Meanwhile, pneumatic actuators, such as the Festo DSBC Series, are used for controlling valves in fluid and gas management systems. Together with sensors, they create a feedback loop of data and appropriate action derived from it, allowing the IoT to interact with the environment in a smart and adequate manner.

Communication and Networking Technologies

Communication and Networking technologies are the backbone of IoT systems, as without their seamless and reliant data transmission and propagation among the devices, platforms, subsystems, and end users. They are essential in the proper functioning of any IoT smart environment on any scale across various applications ranging from smart homes to large-scale implementations like Smart Cities. The chosen technology, communication methods, protocols, and networks significantly impact the scalability, efficiency, and adaptability of IoT systems, so a tailor-made solution is needed to facilitate different scenarios that demand specific capabilities and trade-offs.

Solutions in the localized IoT setups are often utilizing short-range communication technologies like WiFi, Bluetooth, Zigbee, and Z-Wave [50, 17]. In applications with bandwidth-intense requirements, such as camera surveillance, the high data rates of WiFi provide a way to stream this vast amount of data to a processing server. For the needs of low power drain transmission, Bluetooth is a widely employed technology of choice. Often found in wearable devices such as smartwatches, various sports trackers, and personal area networks due to their low power consumption. Zigbee and Z-Wave excel in mesh networking, allowing devices to communicate indirectly by relaying signals through intermediate nodes, which is crucial for ensuring reliable connectivity in environments like smart industrial setups or home automation systems.

These short-range technologies, while reliable for localized deployments, face limitations in range, scalability, and robustness, making them unsuitable for large-scale IoT systems. While there are means to implement a wide-scale solution with short-range communication technologies, such implementations are often fragile due to the nature of their implementation. This is where long-range communication technologies come into play as they were designed to accommodate these needs with appropriate fallback systems to keep the communication up and running.

For long-range communication, Low-Power Wide Area Networks (LPWANs) [8] like LoRaWan and Sigfox are key in enabling IoT applications transmission over vast areas with minimal energy consumption. These are especially valuable in agriculture where the sensors monitoring the soil moisture, various crop conditions, or livestock location are often located and operated in remote areas with limited access to power and infrastructure [2]. Similarly, cellular networks, including LTE-M and emerging 5G technologies, offer higher bandwidth and lower latency, making them ideal for real-time applications such as autonomous vehicles and Smart City infrastructure. For instance, 5G networks can support millions of devices within a small geographical area, facilitating advanced IoT use cases like connected traffic systems, predictive maintenance in utilities, and remote healthcare [41].

The integration of communication technologies in IoT systems is underpinned by protocols like MQTT (Message Queuing Telemetry Transport) [27] and CoAP (Constrained Application Protocol) [7], which are specifically designed for efficient data exchange among devices displayed in Figure 2.2. MQTT is a lightweight messaging protocol that operates on a publish-subscribe model. A central breaker manages the message delivery between the publisher, a device sending data, and the subscriber, devices that have opted to receive data from the given source. This allows devices to receive only the relevant data thus reducing unnecessary network traffic and optimizing communication. This protocol is particularly effective in scenarios requiring reliable message delivery with minimal overhead, such as real-time monitoring systems in industrial IoT.

Conversely, CoAP is a web-based protocol following the client-server model tailored for low-power and resource-constrained devices. Utilizing the Representational State Transfer (RESTful) architecture similar to the HTTP but more optimized for IoT usage by using lightweight methods such as GET, POST, PUT, and DELETE to enable efficient communication. CoAP supports multicast communication which allows multiple devices to receive updates and the same data simultaneously. This comes in handy when one sensor is relaying its data to multiple nodes at once. Additionally, CoAP utilizes User Datagram Protocol (UDP), thus making it ideal for devices with limited processing power or energy constraints.

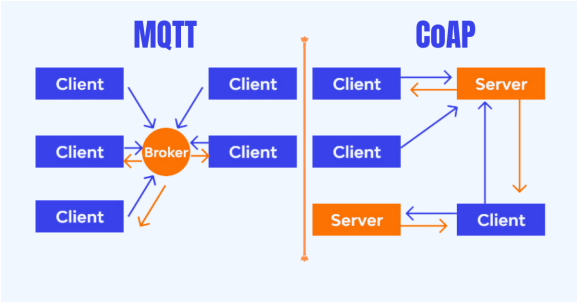

Figure 2.2: MQTT vs CoAP comparison

However, these protocols have their limitations. MQTT is generally efficient due to its lightweight nature, but it can struggle in high-throughput scenarios where rapid message delivery and processing are required [27], making it less suitable for applications requiring extensive data processing or complex interactions[45]. CoAP is well-suited for constrained environments with low power and limited bandwidth[45]. It may lack robustness in handling diverse and large-scale data streams. These constraints necessitate hybrid approaches in demanding IoT applications, combining the strengths of multiple protocols to achieve a balance of efficiency, functionality, and scalability. In high-density environments like Smart Cities, communication technologies face additional challenges [50], such as network congestion, which can result in increased latency and data loss.

Another essential piece of technology is the RFID (Radio-Frequency Identification) [47], which enables efficient wireless data transfer and device interaction. RFID utilizes tags embedded with information that can be wirelessly read by the RFID reader using radio waves. There are either active or passive tags, with the active ones having their own power source and the passive ones relying on energy emitted by the RFID reader. This allows for high-speed scanning, identification, and tracking without direct line-of-sight, which enables rapid data capture in a variety of environments.

NFC (Near-Field Communication) [40] is a subset of RFID which operates at a shorter range and higher frequency with bidirectional communication between devices that are within a few centimetres. It is designed to facilitate the secure exchange of data via encryption and authentication protocols, ensuring the safety of these interactions. NFC, in comparison with RFID which is used mainly for tracking and tagging, enables more interactive behaviour such as device pairing, transferring of a small data amount, and access authentication. In a similar fashion to RFID tags, NFC tags can operate without their dedicated power source as they utilize the electromagnetic field to transfer data.

Data Processing and Storage Technologies

Data processing and storage technologies are critical to the effective functioning of IoT systems, ensuring that the vast amounts of data generated by connected devices are managed, analyzed, and utilized efficiently. Cloud computing plays a central role in this ecosystem [17], offering scalable storage and powerful analytics capabilities through platforms like AWS IoT and Microsoft Azure IoT. For example, in smart agriculture, cloud computing enables the aggregation of data from multiple sensors monitoring soil conditions and weather, providing farmers with actionable insights for optimized crop management. However, reliance on cloud computing introduces challenges such as latency and the need for constant connectivity, which can be problematic in remote or time-sensitive scenarios.

To address these issues, the aforementioned edge computing [43] is utilized. Similarly, fog computing facilitates an efficient data flow between edge devices and the cloud, enabling applications like smart grids to process energy consumption data locally while centralizing broader analyses for system optimization [6]. Furthermore, big data analytics enhances the value of IoT systems by transforming raw data into insights [50].

Tools like Hadoop and Spark [42] are widely used to analyze large datasets, enabling predictive maintenance in industrial IoT. For example, General Electric employs big data analytics to monitor turbines and engines, identifying potential failures before they occur, thereby reducing downtime and costs [17]. As IoT continues to evolve, emerging trends such as AI-driven data analysis and quantum computing hold immense potential [2]. AI models can identify patterns in sensor data to improve energy efficiency in smart buildings, while quantum computing promises to solve complex optimization problems in logistics and supply chain management faster than traditional computing methods.

Security and Privacy Technologies

Ensuring security and privacy is a critical challenge in IoT systems [16], as they involve vast amounts of sensitive data exchanged across interconnected devices and networks. Encryption protocols such as AES (Advanced Encryption Standard) and TLS (Transport Layer Security) play a vital role in protecting IoT communications. For example, TLS secures data-in-transit by encrypting information exchanged between IoT devices and cloud servers, safeguarding it against eavesdropping and interception. In healthcare IoT, this is essential for transmitting sensitive patient data securely, ensuring compliance with regulations like GDPR [26].

Authentication and access control mechanisms are equally important for preventing unauthorized access [1]. Multi-factor authentication (MFA) combines multiple verification steps, such as passwords and biometric data, to strengthen security. Additionally, role-based access control (RBAC) assigns permissions based on user roles, limiting access to sensitive operations. Blockchain-based identity management is emerging as a solution to

enhance device authentication by creating tamper-proof digital identities, particularly in large-scale IoT deployments like smart grids [41].

IoT systems also require robust mechanisms for threat detection and mitigation [43]. Machine learning algorithms are increasingly used to detect anomalies in network traffic, identifying potential threats such as DDoS attacks and spoofing attempts. For example, anomaly detection systems in industrial IoT can analyze real-time sensor data to identify suspicious patterns that may indicate cyberattacks, allowing operators to take preventive action. Despite these advancements, balancing privacy with functionality remains a challenge [26]. Regulations such as GDPR enforce strict data privacy requirements, compelling IoT developers to anonymize user data while maintaining its utility for analytics and decision-making. By integrating encryption, robust authentication, advanced threat detection, and compliance with data protection regulations, IoT systems can address the dual challenges of security and privacy, ensuring trustworthiness and resilience in diverse applications.

2.3 Usages of IoT

Although Kevin Ashton coined the term "Internet of Things", the first known IoT implementation is traced back to the early 1970s with the CMU CS Department's Coke machine [11]. Frustrated by empty or warm bottles after long walks to the machine, students equipped it with microswitches to monitor each slot. These switches tracked whether the slot was empty, recently loaded, or cold after a three-hour timer. The machine was connected to a PDP-10 computer, enabling staff to remotely check its status. Over time, it became an internet icon, accessible globally through the Finger protocol, allowing users to query its status from any networked computer.

The ambitions with this technology reach far beyond the size and scale of a mere Coke Machine connected to a terminal displaying some data. These days, the usage and integration of IoT find their way into many sectors with varying applications. The main trends regarding IoT use are the creation of Smart Homes [35], improvement and modernization of healthcare [17], Smart Agriculture [17], Industry 4.0 [4], and the creation and development of Smart Cities [10].

The homes can easily be made into Smart Homes as there is no solid definition for what constitutes a Smart Home [35]. The first steps to Smart Homes often, at least mine did this way, start with a few smart devices, such as automated light control or an SMS-operated garage door, making our life easier. These first steps can be taken by the average Joe while other more advanced like smart security systems or smart heating require more skill and knowledge. However, after installing them, the user interactions are really simple, often letting us utilize our smartphones as the main control hub. Thus letting a wide range of users interact and use them to their full potential.

Healthcare is another sector where the IoT can be applied to better and streamline the daily workings of it. Allowing for far greater and more accurate oversight of patients while lowering the load on each staff member. Smart monitoring bracelets [17] or integrated systems across different types of medical care facilities can help in making faster and more accurate observations, allowing the medical personnel to provide more well-fitted medical care and potentially catch underlying conditions that might otherwise go unnoticed.

For Smart Agriculture, there is a range of sensors that help monitor different aspects, such as soil moisture, weather conditions, and the state of the plants, enabling more precise farming to optimize yield and resource usage [17]. In a similar fashion in the industry setting, fitting the machinery with sensors [4] that monitor the vibrations and inner workings

of the machine to alert the operator to the failure. These sensors can predict potential failures, saving valuable time and resources.

The current urban performance of cities does not solely depend on the growth and improvement of the physical infrastructure, as it can not sustain the expansion rates alone. There is a need for a solution which provides a way to utilize the resources and systems already present in a better and more effective way while not being invasive to existing infrastructure [10]. The concept of „Smart City“ has been introduced as a proposed solution or, rather, a framework to be applied to cities and their future planning, nourishing and supporting urban production factors.

These factors [10] include human capital, the skills and education of the workforce, social infrastructure such as healthcare, education, and social services essential for city inhabitants, economic infrastructure for the business and industry systems, physical infrastructure including transportation, utilities, and housing suitable to the changing needs, and digital infrastructure like the Information and Communication Technologies networks (ICTs) [25] and overall digital connectivity. Digital infrastructure, in particular, has become critical in Smart Cities, as ICT enhances the management and coordination of other urban resources.

2.4 Smart Cities and their Conceptualization

A general definition of a Smart City is an urban area utilizing advanced technologies, particularly the ICT and IoT, to improve infrastructure, optimize resource use, and enhance the quality of life for its residents [10]. The ever-growing trend of population concentration in large cities provides many opportunities while creating challenges that need to be met if they are to be successful. These challenges can be managed by the integration of ICT and IoT technologies into various urban systems aiming to achieve greater efficiency in various areas such as public transportation, energy and waste management, public safety, and fostering sustainable, resilient, and low-impact urban environments [18].

The concept of Smart Cities is, however, multidimensional and extends beyond just technologies. The governance and citizen engagement are as critical as the technical infrastructure. The governance consists of establishing and upkeeping policies, regulations, and frameworks that support technological innovation while not overlooking issues like data security, privacy, and public accountability, which must be addressed [29].

Citizen engagement plays a crucial role in ensuring that Smart City initiatives are designed and kept up to date with the needs and preferences of its inhabitants. For proper citizen engagement and interaction, the necessary technology must be made as accessible and convenient as possible, enabling and encouraging the residents to participate actively in urban development and management [26]. This citizen access-driven approach, which recognizes multiple interconnected fields within the Smart City, is essential in creating sustainable and adaptive urban spaces that blend technological advancements with inclusivity and social well-being. By involving citizens and governance structures, Smart Cities foster a collaborative environment where technology serves as a tool for achieving broader social and environmental goals [12].

The Smart City infrastructure relies on a couple of core pillars that enhance urban functionality, sustainability, and livability. Those are transportation and mobility [2], energy and utility infrastructure [26], public safety [29], and environmental management [10]. Together, these pillars create a resilient foundation for sustainable urban growth, addressing key urban challenges through integrated smart technology and thus will be the main talking points throughout this chapter.

Role of Information and Communication Technologies

Information and Communication Technologies (ICTs) form the structural backbone of Smart Cities utilizing real-time data collection, analysis, and automation to enable the integration and optimization of urban services [26]. This allows for a vast amount of data collection from various sources, such as sensors, cameras, and various IoT devices, together facilitating a coordinated response to various situations and problems that can occur on a daily basis in a busy city, like traffic congestion and fluctuating load on public transport, energy consumption and potential power outages, and effective waste management and disposal.

An essential component of ICT in Smart Cities is the broad and high-quality deployment of advanced communication networks such as broadband, 5G, and fibre optics, which support high-speed data transmission and connectivity across urban areas [18]. These networks serve as the city's central nervous system, sending fast signals to attend to each situation arising through real-time monitoring and management of the city infrastructure, fostering precise and agile decision-making and rapid response to emergencies.

ICTs major benefit, which they provide through monitoring and data-driven insights gathered through them, is energy and water use optimization, greatly boosting sustainability and reducing unnecessary waste [12]. Other useful utilities include the upkeep of public places via automatic watering systems, light pollution-reducing street lighting features, monitoring of the fullness levels in bins, and providing overwatch for citizens to provide a safer and more comfortable living experience.

2.5 IoT in Smart Cities and Smart Government

The Internet of Things (IoT) is the building block in the creation of Smart Cities as this technology enables the integration and coordination of devices to improve the quality of life and urban functionality [17]. In the context of Smart Cities, IoT connects various IoT-enabled devices, such as sensors, cameras, and smart meters to gather and transmit data about a wide range of aspects of city infrastructure, including traffic, energy consumption, environmental conditions and factors, and public safety or recognizing medical emergencies.

One of the main applications of IoT in Smart Cities is traffic management. The connected sensors and cameras monitor the situation and collect data on vehicle flows and potential accidents, pedestrian movement, and public transportation usage [50]. The data and conclusions drawn from it allow the Smart City to adjust the traffic signals or electronic variable speed signs, reroute traffic in case of accident or sudden congestion forming, and provide real-time accurate data to prevent further congestion and lower the accident rate thus improving both the effectiveness and safety.

In a similar fashion utilizing ICT networks connected to the power grid, IoT-enabled smart grids are created [41]. These smart grids help monitor energy consumption across the city regions and districts, balancing the demand without waste while lowering the chances of power outages or overloads, and utilizing various renewable energy sources according to the available supply. This greatly enhances energy efficiency saving finances and the environment being an overall improvement.

IoT also enhances the possibilities of environmental monitoring, with sensors measuring air quality, noise levels, pollen, and water quality across the city [29]. This data aids the city in optimizing the living conditions by reducing pollution through the aforementioned traffic control and other means, responding to environmental threats such as fires, floods,

or contamination spills, and overall maintaining a healthy and clean urban environment for the residents.

Smart Government is another critical application of IoT in the context of Smart Cities [29]. Focusing on enhancing governance through technology-based decision-making and active citizen engagement. These new technologies allow for better monitoring and analysis of the events happening in the city, thus providing a base to tackle them more efficiently. Interconnected public administration systems, automated service platforms, and intelligent data-sharing frameworks are just some of the ways IoT supports Smart Governance [2].

Chapter 3

Visual Programming Languages and End-user Development

As demonstrated in the previous chapter, the IoT gives us immense possibilities. However, the main goal is to take the full advantage of it by providing the end-user with a platform packed with utilities and easy-to-use tools. Thus equipping the user with all necessary parts to interact with and take advantage of IoT environments. For the full effect and user coverage, the main focus should be on the platform choice and ways of interaction that are the most handy to the user while posing as little entry barrier as possible.

The most universally used platform among device users is the mobile phone. Mobile phones are the most universally used devices among users worldwide. As of 2023, approximately 78% [20] of the global population aged 10 and over owned a mobile phone, surpassing the percentage of individuals who use the internet. In the United States, 97% of adults own a cellphone [36] of some kind, with 90% owning a smartphone. This widespread adoption underscores the mobile phone's role as the most prevalent platform among device users globally. Due to this massive representation, the platform we mainly focus on should be mobile devices. However, due to their often small screen, they do not have the ability to facilitate the conventional means of programming and routine definition. This can be achieved with the help of correctly chosen or developed EUDF tools to accommodate the user's needs.

Parametrization plays a crucial role in allowing the user to gain full control of the smart environment. It is often also the part of the environment that most users interact with as the smart routine can be created once or via template and reused for various similar use cases. Let's take a smart greenhouse that is equipped with sensors and provides automated watering capabilities controlled via parametrization in a phone app. The application has an overview of the sensor's current values, based on which the user can create conditioned actions to control routines. The gardener can select the desired soil humidity for their given plant, and the watering system takes care of the rest while providing the gardener with useful data. For this to work we need an easy and effective way how to allow the user to enter and modify these parameters.

Currently, one of the most commonly used solutions is the If This Then That (IFTTT) [46]. If This Then That is a web-based trigger action programming platform designed to simplify automation by allowing users to connect applications, services, and devices through simple "if-then" logic statements known as applets. Launched in 2011, IFTTT has gained popularity for its user-friendly interface, which enables non-technical individuals to

create automation without requiring programming knowledge with just their smartphone. For example, users can automate tasks such as saving email attachments to cloud storage, turning on smart lights upon arrival at home, or receiving weather alerts. The platform's strength lies in its accessibility and extensive integrations [23], supporting hundreds of services and IoT devices, which makes it a versatile tool for personal productivity, smart homes, and social media management.

Despite its advantages, IFTTT also has notable limitations [30]. While it excels at providing a straightforward way to connect services, as seen in Figure 3.1, its functionality is restricted to basic workflows and lacks the complexity needed for advanced automation tasks involving multiple conditions or custom logic. This limitation reduces its applicability in professional IoT environments requiring more robust solutions. Additionally, IFTTT's reliance on third-party APIs means its applets are dependent on the continued availability and reliability of these external services [46]. Changes or discontinuation of these APIs can disrupt automation, making the platform less stable for critical use cases. Its simplicity and reliance on external services highlight the need for more advanced platforms in complex IoT ecosystems.

Figure 3.1: Simple trigger in ITTT

Another popular solution is Zapier [51]. It is a web-based automation platform which lets the user connect web applications into automated workflows, known as Zaps, without any coding skills. It relies on events set as triggers in one app and actions represented by various actions in another app to streamline repetitive processes. Its integration with thousands of apps like Gmail, Google Sheets, and Slack makes it a popular choice in business operations or marketing as it helps to reduce the manual effort needed.

Solutions like Node-RED or customized IoT platforms offer greater flexibility and control, making them better suited for professional or large-scale applications requiring granular logic and enhanced security. More focus will be paid to how to tackle these challenges and meet the needs that arise with them as well as go over the current state of the VPLs.

Visual Programming Languages (VPLs) is a term associated with a range of forms and utilities, all sharing one main common goal. Abstracting the code representation and creation from the traditional code written in text into a simpler visual form [23]. Utilizing visual elements such as blocks, icons, or flows, thus shifts the major load off the user and provides an experience with a lower entry barrier. This visual representation of programming logic allows the user to connect predefined modules, perform actions, and manage data flows through various plays on the simple drag-and-drop interface. Such an approach to the code input from the user means the possibility of simplifying complex programming tasks, allowing users with little to no technical background to create routines or simple programs, thus letting them interact with the IoT. This way is often more intuitive and accessible which is essential in the success of Smart Cities. The primary focus is parametrization as this is the part with which the user interacts the most often.

3.1 Historical Context and Evolution of VPLs

The VPLs have actually quite an extensive and long history, within the scope of the rather short overall history of Information technology. Their beginnings date back to the 1960s and 1970s. The pioneering tools that set the tone for VPLs were basic Flowcharts and Sketchpad [48]. These early graphical abstractions allowed for a visual representation of computational processes. The Sketchpad [44] was a leap forward in particular as it let the user directly manipulate the visual elements via a graphical interface, letting them create various shapes and then interact with given shapes, setting the stage for further improvements and innovations in this field.

VPLs picked up on popularity in use and later on the widespread adaptation within the scientific and engineering community in the 1980s with the introduction of LabVIEW. Enabling the user to design programs through the GUI allows the user to focus more on the development rather than the implementation. This was proven by the observational study from the Measurement Technology Center (MTC) [48] in which two research teams were both tasked with the same requirements and provided the same resources with one team using the VPL LabView and the other standard C language. After the elapsed time, the VPL team fared way better than the stated requirements, whereas the latter team did not even meet them, let alone exceed them. Proving that the concept VPL has fulfilled its aim of being more accessible and simpler while providing the desired utility. This was a clear go sign for VPLs with further research and resources being allocated for them.

The widespread popularity among other fields did not come until the late 1990s and early 2000s when the trend of VPLs tailored for educational purposes came to fruition. The most notable one was the creation of Scratch by MIT [39]. It was designed to teach young students the fundamental concepts of programming while not burdening them with complex syntax needed for correct code execution achieved with a simple block-based VPL.

Development and growth of the VPLs did not stop there and, in recent years, evolved to more closely meet the needs of web and IoT applications. Tools like Node-RED and Google Blockly have been developed to satisfy this demand. Providing accessible, flow-based interfaces for the creation of IoT workflows and web-based programs. Node-RED lets the user connect and control IoT devices with ease by linking nodes in a flow-based environment [5]. Its universal application and simple interface propelled it to the forefront of swift prototyping of IoT solutions. The Googles take on VPL, Blockly [14], is by its telling name a block-based VPL that provides a visual abstraction over many programming languages all under one VPL interface. This bridges the gap of various syntaxes across

these, often vastly different, coding languages while accommodating a range of applications ranging from education to web development. Doing all this while retaining most of the utility each one provides. These VPLs will be described in greater detail in the subsequent sections.

3.2 Comparison of Text-Based and Visual Programming

As we have mentioned, the main difference in which the VPLs stand out over the traditional text-based programming languages is their visual representation of user interaction and presentation of the data on UI. VPLs prioritize accessibility and ease of use above all else, facilitating a quick and easy code creation experience despite sacrificing or complicating a few edge use cases [34]. In traditional text-based programming, the actual coding phase is prefaced with often complex and lengthy learning procedures necessary for the creation of even the simplest usable applications. This creates a steep learning curve that often deters most users from partaking in such activities. Not having the necessary coding knowledge leads to the creation of faulty code with the need for tedious debugging. These factors result in unnecessarily long and frustrating development.

The Visual Programming Languages provide the user with predefined visual elements. The graphical elements, such as blocks, diagrams, or nodes in flowcharts, snap and connect together in a seamless and logically coherent way [39]. They represent the program's logic in accessible and intuitive while eliminating syntax errors. This feature lowers the cognitive load on the users, allowing them a more concise and comfortable workflow [23]. This visual clarity is particularly beneficial in the context of IoT where often complex applications with a multitude of interconnected devices and data flows are implemented [5]. The VPLs can help cut down the time between picking up a new technology and being able to create usable outputs by it.

Ultimately the visual approach of VPLs opens up the doors of programming utility to a wider audience with varying skill levels and technical backgrounds [5]. This accessibility is especially relevant in collaborative environments like Smart Cities, where technical and non-technical stakeholders need to work together on IoT projects and policy management.

3.3 Overview of the Existing VPLs

While there are many different classifications for VPLs based on their target users, interaction or execution model, and other criteria, I find the most relevant for this thesis to be the representation type. The main types based on the representation division are the Block-based, Dataflow, and Diagrammatic VPLs described in more detail below.

Block-based VPLs

Block-based VPLs such as Scratch [39] or Blockly [14] operate on a system of predefined and user-defined blocks that snap together in a puzzle-like manner, enforcing syntactically correct code structures. Each block represents a specific function or operation, such as loops, conditions, mathematical operations, or user-defined functionality, with the connections between the blocks defining the program's flow which is then executed per the VPLs rules. In Figure 3.2 a simple program created in the block-based VPL Blockly is shown to demonstrate the functionality. The program is visually constructed by rearranging blocks

and filling in the conditions that define the program logic. Blockly then generates text-based code in JavaScript that can be executed. The users can also pick Python, PHP, Lua, or Dart as their target text-based programming language, making it a versatile choice.

One of Blockly's key strengths is its intuitive user interface [14], which simplifies programming for beginners while maintaining the flexibility required for more advanced users. The drag-and-drop mechanism eliminates syntax errors, and the visual layout clearly displays the logical flow of the program, making it particularly well-suited for educational environments and IoT applications. Additionally, Blockly's ability to integrate into other platforms as a framework allows developers to create customized VPL environments tailored to specific needs.

However, Blockly has some limitations in user interaction. For example, as programs grow in complexity, the workspace can become cluttered, making it difficult for users to navigate and manage large blocks of logic [14]. This visual complexity can be a barrier for advanced applications requiring intricate workflows [30].

Blockly enables users to define their own functions through custom blocks [14], which can encapsulate specific logic or operations for reuse. Users can create a function block by specifying input parameters and defining the internal logic using other blocks. This functionality supports modular programming practices, making it easier to manage and scale programs. For example, in an IoT application, a user could define a function block to calculate temperature thresholds for activating a cooling system. The creation of custom blocks in Blockly can also be achieved via manual implementation using JSON files or a JavaScript API to provide better control to users with higher technical knowledge. In the block, the name, type, input parameters, flow connections, and colour are defined, allowing for this block to be then loaded and used in the program. These custom blocks can then be reused in multiple workflows, reducing redundancy and improving program clarity.

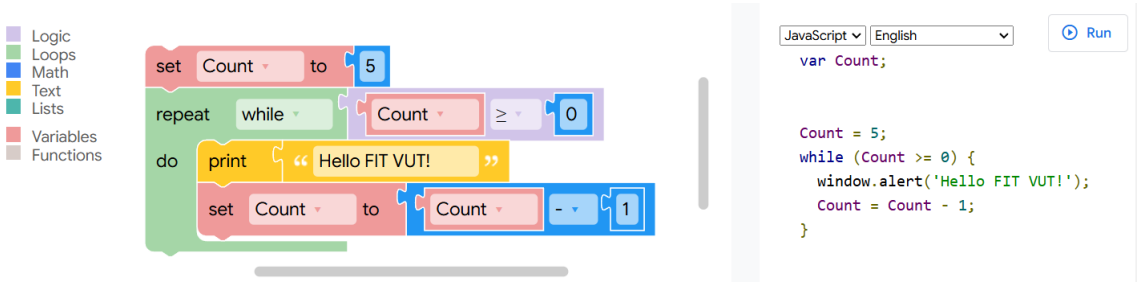

Figure 3.2: Simple test program in Blockly

Dataflow VPLs

Dataflow VPLs' main focus is the visual representation of the data flow between nodes or blocks [5]. As you can see in Figure 3.3 demonstrated on a simple Node-RED program, each node or block performs a specific operation, and each connection between them indicates the path of the data being processed, thus representing the program logic.

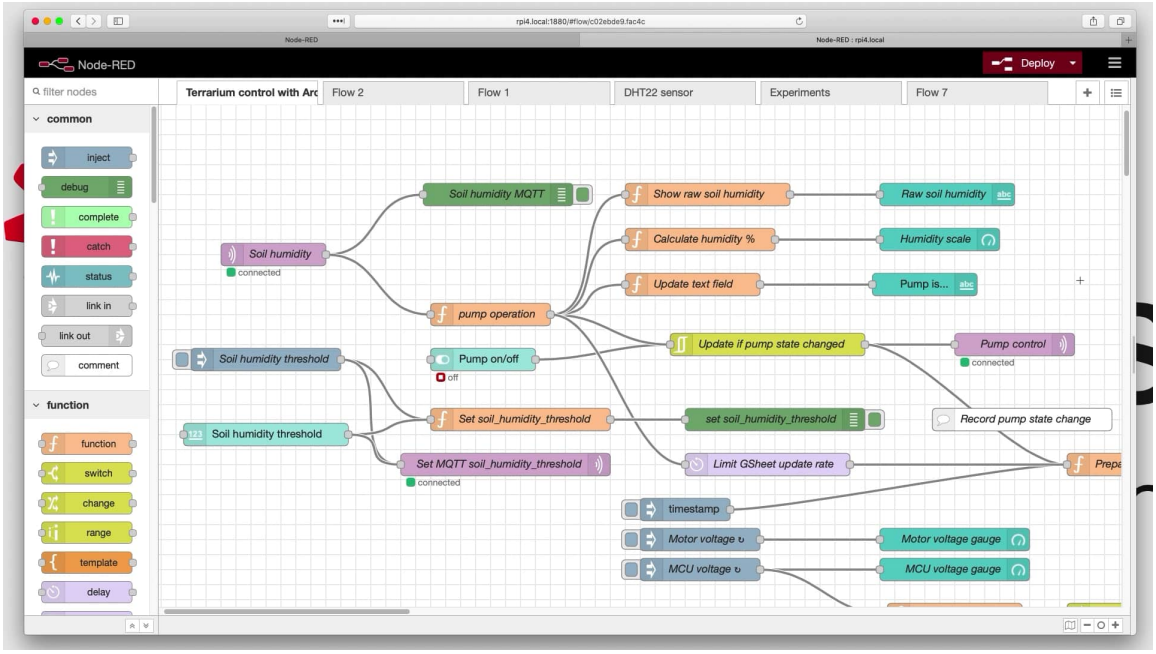

Figure 3.3: Example Node-RED program

Node-RED [5] is an open-source visual programming tool designed for wiring together hardware devices, APIs, and online services in IoT systems. Developed by IBM's Emerging Technologies group, Node-RED provides a flow-based development interface that allows users to create workflows by connecting nodes, each representing a specific function or device. It operates on a browser-based editor where users drag and drop nodes onto a workspace, link them together to define the data flow, and configure each node's properties. These flows are executed by a lightweight runtime built on Node.js, which makes Node-RED highly extensible and suitable for resource-constrained IoT environments.

The logic revolves around event-driven flows, each node performing a specific task such as receiving, processing, and sending data. The nodes are connected by lines or wires, each representing the flow of messages, visualizing the logic of the program. For instance, a flow may consist of an input node, such as data from a temperature sensor, a processing node, like applying a threshold rule and evaluating the data, and an output node for sending an alert if the threshold is exceeded. This modular approach makes it intuitive to design and debug IoT applications.

Node-RED excels in accessibility for users with basic programming knowledge, offering a drag-and-drop interface that simplifies complex tasks. Its node palette provides pre-configured nodes for common operations, such as connecting to MQTT brokers, making HTTP requests, or integrating cloud services [5]. However, despite its user-friendly interface, Node-RED can present challenges for non-technical users, particularly in understanding how to configure nodes or manage data types effectively.

It allows users to define their own custom functions using its function node, which supports JavaScript programming. This flexibility enables users to implement bespoke logic that goes beyond the capabilities of pre-configured nodes. For example, users can write a function that dynamically adjusts an IoT system's behaviour based on real-time sensor data or custom algorithms. The function node also supports debugging by displaying outputs and logs, enhancing the development process. The visualization of custom logic in

Node-RED is handled through its flow-based interface, which represents each function as a node within the workflow [5]. This makes it easier for users to see how custom functions integrate with other components and how data flows through the system. Additionally, Node-RED's integration with dashboards allows users to create interactive visualizations, such as charts and status indicators, to monitor and control their IoT systems in real-time [50]. This capability is especially valuable in applications like smart homes or industrial IoT, where live data representation is crucial for decision-making.

Diagrammatic VPLs

Diagrammatic VPLs use structured diagrams, such as flowcharts, to visually represent program logic and, most importantly, the system behaviour [48]. In comparison to dataflow VPLs, which focus mainly on the data flow, diagrammatic VPLs focus more on the description and representation of the system. These languages find their main application in domains with the need for modelling, simulation, and analysis of complex systems. A notable example is Simulink [38], a diagrammatic VPL mainly used in engineering and industrial applications.

Simulink allows users to create block diagrams to model, simulate, and analyze dynamic systems, such as control systems or signal processing workflows. It offers a way to solve equations numerically using a graphical user interface rather than requiring code. This approach simplifies the representation of complex interactions by using visual elements that can be intuitively understood and manipulated. In Figure 3.4, a program simulating a wind turbine system is demonstrated. Diagrammatic VPLs are especially valuable in professional settings, where precision and clarity are essential for tasks like prototyping or verifying system behaviours before deployment.

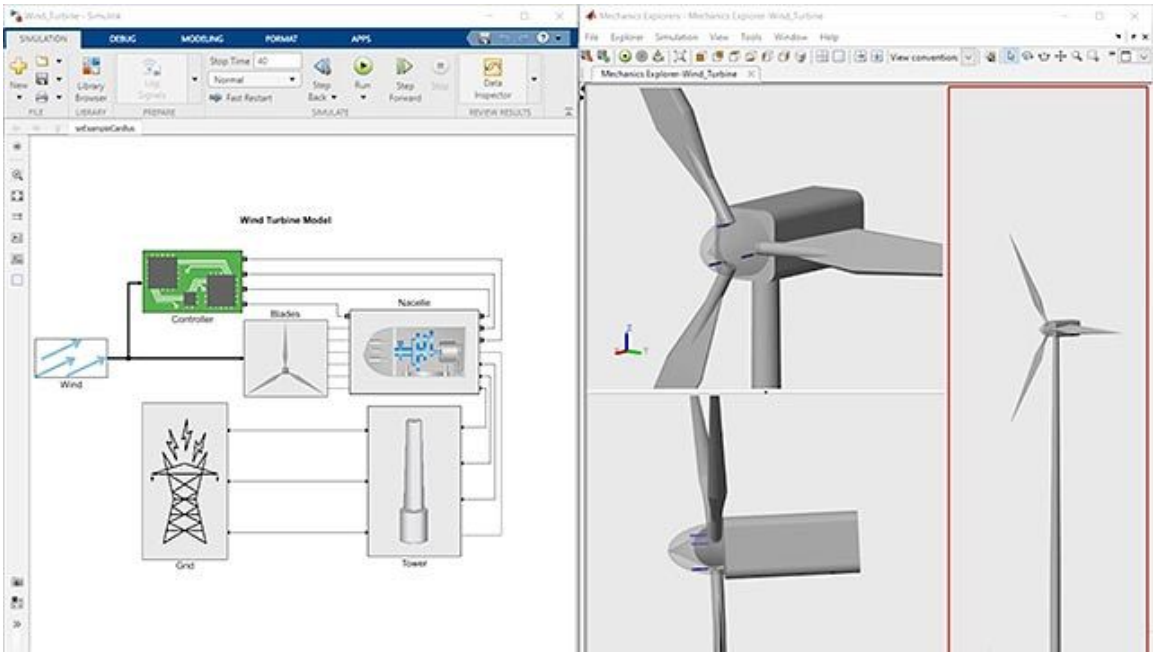

Figure 3.4: Example of Simulink program

3.4 Designing User-Friendly Solutions for Custom Function Workflows

User-friendly solutions for custom function workflows in IoT VPLs are essential to accommodate the creation of complex IoT systems accessible to as many users as possible. The IoT will find its way more and more into our everyday lives and thus must accommodate a way for the user to define their own procedures.

The VPLs used in these systems must offer intuitive interfaces that allow users to easily define, manage, and visualize custom functions without the need for extensive programming or prior briefing [13]. In this context, user-centred design principles, such as simplicity, consistency, and interactivity, are critical focus areas essential for a long-term user-oriented solution. An effective IoT VPL should implement these principles with the user's varying programming skill levels in mind. Ensuring the VPL is simple enough to grasp with minimal prior information needed, but not sacrificing the utility essential for the more skilled users to find the VPL useful. Utilizing the standardized ways of user interaction that the user is used to form his other daily usage of this mobile such as press and hold for further interaction or drag and drop utility.

By prioritizing usability and accessibility [39], the VPLs can create a smooth interaction with the interface, bridging the gap between the technical complexity and various user needs and providing more effective and inclusive IoT usage and development.

An important aspect that must be carefully considered is the main targeted platform, as it heavily influences the user's way of interacting with the VPL. Due to Pocketix being developed as a mobile VPL IoT-enabled solution, the designing principles for mobiles will be mainly examined. For mobile development, designing user-friendly VPLs involves addressing specific challenges such as smaller screen real estate, touch-based interaction, and the need for on-the-go functionality [13].

An effective solution must implement responsive layouts and streamlined workflows to enhance usability on mobile devices. Careful balance must be maintained of the information provided, with the issue of over-cluttering the screen and overwhelming the user's mind. Utilising collapsible menus or zoomable workspaces to better organise and visualise the workspace could partially solve the issues. Additionally, supporting multi-touch gestures for connecting or rearranging blocks can enhance the interactivity and intuitiveness of the interface. In the following chapter 4, the current state of EUDFs and the ways a user can define their own procedures in different VPLs will be examined with the aim of evaluating different approaches and their potential application in developing the solution for this thesis.

Chapter 4

Analysis of the Current Implementation and Existing Frameworks

As mentioned throughout the thesis, the proposed solution should cater to the target audience of users with little to no technical knowledge. Thus, it must be simple enough as not to overwhelm the user while providing them with a rich repertoire of functionality.

The solution will be building on top of an already existing visual editor supporting a Pocketix block language, with minimal support for user-defined functions already implemented. The VPL Editor will be further integrated into the RIoT system, providing a fuller and more refined experience. This chapter aims to go over the current solution and to identify necessary changes to accommodate the support for UDFs, while modifying the logic to better support integration into the RIoT smart system.

RIoT, a Real-time Internet of Things, which started as a bachelor's thesis by Bureš Michal [9]. RIoT is an advanced information system designed for real-time IoT data processing and state monitoring, enabling users to define custom key performance indicators (KPIs) to track and optimise system behaviour. It encapsulates a smart environment, allowing the user to manage various smart devices [22], create dashboards [33] to visualise the obtained data, and create procedures to execute various routines on said connected IoT devices.

The goal of this analysis is to identify the main key areas and aspects that need to be reworked or developed, with regard to the creation and use of reusable user procedures within the RIoT integrated VPL Editor. Currently, the RIoT system is being further developed and optimised in joint efforts by students under the supervision of Ing. John Petr under the umbrella of the Pocketix organization[1].

4.1 Pocketix VPL

The VPL Editor operates on the Pocketix VPL, described in more detail in both the thesis by [37] and voucher [21] by Petr John. The Pocketix VPL establishes both the **metamodel** and the **program model**, while also dictating certain editor-specific features and limitations.

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

The metamodel outlines the structural constraints and valid hierarchical relationships within the VPL. It specifies permissible parent-child configurations, preventing invalid constructs, such as simple commands containing child nodes. Additionally, it also captures visual attributes like block colours and icons for representation in the editor.

Two primary statement types, simple and compound, are defined in the metamodel and further expanded on by the thesis of Bc. Lukáš Podvojský [37]. Simple statements, represented by the `LanguageCmdStatement` type, are basic commands with parameter lists, typically of types `Number`, `String`, or `Boolean`. Compound statements extend the base type `LanguageStatement`, enabling conditional logic, which allows for constructs such as `While`, `If`, or `Else`.

The **program model** serves as the object representation of VPL programs, facilitating evaluation and REST API communication. Each program instance is represented by a `Program` object, containing a single `Block` that holds a list of statements. These statements are further categorised as `AbstractStatement`, `CompoundStatement`, or `Command`. Compound statements mirror their metamodel counterparts, supporting conditional execution, while commands are individual executable statements with parameter lists.

Although the structure may seem redundant, it is deliberately designed to support future extensions, such as program-scoped variables, without modifying the core model. This flexibility ensures a scalable and maintainable program representation within Pocketix VPL.

4.2 Current State of VPL Editor

The current VPL Editor[2] is the product of Bc. Lukáš Podvojský described in his thesis [37]. It provides a well-structured solution allowing for the creation of programs or procedures and serving as a base for future additions or modifications. The editor is implemented using TypeScript and the lightweight Lit web components library, while the project is configured to be published as an npm package containing all the necessities for integration and use in different projects.

The current state of the VPL Editor, as seen in Figure 4.1, consists of interactive editors and editor controls placed above them. Controls allowing the basic interaction for variable and procedure management, and import/export of the JSON program file. The editor consists of two windows, the graphical editor (GE) and the text editor (TE). A preferred display mode can be selected, allowing for either just the graphical editor view, just the text editor view, or a side-by-side combination of both, as seen in Figure 4.1.

Both of the editor views display the contents of the program block in their respective format, while keeping the contents synced between the two. This means that any block constructed via the graphical editor will automatically translate to the text editor, and changes made to the text editor, such as changing values or fixing expressions, translate back to the graphical editor. Thus facilitating an intuitive solution in line with the overall project's design.

[2] https://github.com/pocketix/vpl-for-things

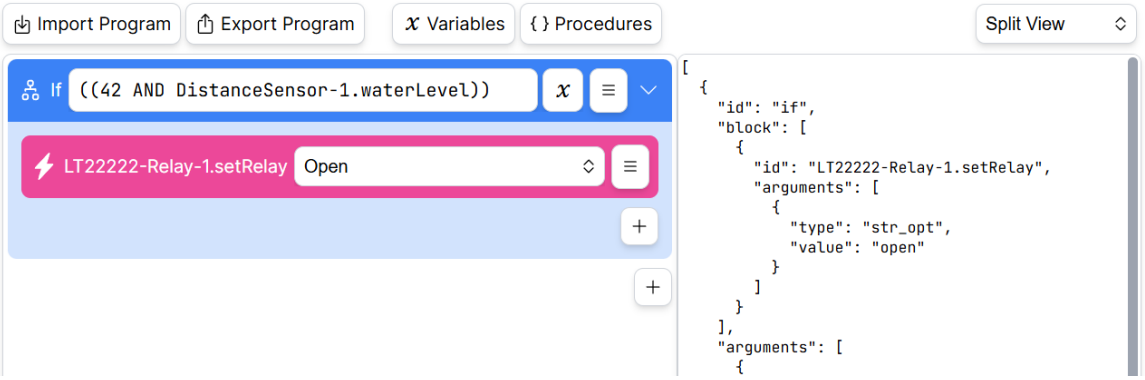

Figure 4.1: Original VPL Editor with simple program displayed in graphical and text view

The VPL program is constructed in the editor using different statement blocks selected by the user via a click-based interface or direct editing in the text editor view. Blocks can be added through the `+` button in the graphical editor view, seen in Figure 4.1, which opens the `Add New Statement` modal featuring two tabs, both of which can be seen in Figure 4.2. The two tabs are the basic statements, with all the editor-native blocks, and the device statements, offering the user all available devices and their respective commands.

Each block has an icon, a name, optional variables or arguments, and a hamburger icon opening a menu for block manipulation. Clicking on the burger menu displays options to either move the block up and down in the body or to delete the block, which removes it from the body but does not delete the block as such from the program. A small arrow on the right side of parent blocks allows for their child blocks to be hidden, helping with the visual clutter when creating or managing more complex programs.

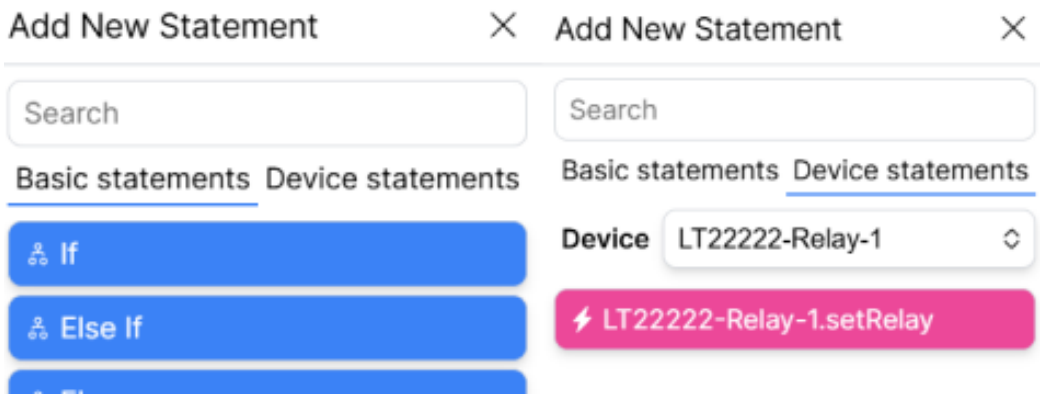

Figure 4.2: The original Add Statement selection with basic and device blocks

The internal representation of the program is represented as a JSON file seen in the listing 4.2. The program file consists of two main parts. The first part is `blocks`, which is an array of statements adhering to the internally defined logic and paradigm representing the body of the program, which is displayed one-to-one in the TE and in its VPL representation in the GE. The latter being the header, consisting of `userVariables` entry, holding the variables defined in the program, and `userProcedures` entry, holding the name and body of the custom user procedure defined in the program. When the body block is being parsed and an ID which is not defined in the basic blocks is found, the `userProcedures` in the

header are parsed, and in case of a match on ID, the body of the given procedure is inserted in place of the parsed ID.

Listing 4.1: JSON representation of the Program structure

```
{
  "header": {
    "userVariables": {},
    "userProcedures": {
      "NewProcedure": [
        {
          "id": "Doorbell-1.takePicture"
        }
      ]
    }
  },
  "block": [
    {
      "id": "NewProcedure"
    }
  ]
}
```

At the moment, the user procedures act more like standalone programs from which the user can build more complex programs with little reusability potential. The limitations in reusability stem from the hard-coded device statements in the user procedures body, making this implementation of UDFs unsuitable for repeated use in even slightly different circumstances. Nor does it support the idea of UDFs being used in a different environment.

An existing user procedure can be added to the program via the `Add statement` button, appearing among the basic blocks in the statement selection. To create a user procedure, clicking on the Procedures button brings out the modal seen on the right in Figure 4.3. Clicking on the `+Add` button brings out the `Add Procedure modal`, seen on the left part in Figure 4.3, where the user defines the procedure name and visual aspects of the block, adding it to the list of available procedures in the procedures modal.

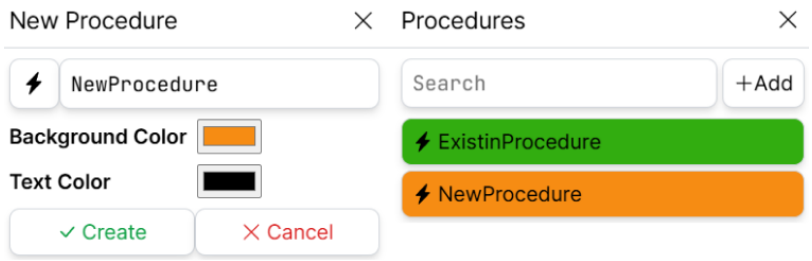

Figure 4.3: The Procedures creation modal (left) and added user Procedures modal (right)

Clicking on any of the defined procedures listed in the procedures modal opens the selected UDF in the editing modal. The procedure modal loads the name, colour, and body of the procedure, as seen in Figure 4.4, showcasing a simple UDF consisting of basic and device statement blocks. At the moment, both the visual aspects and the functionality of creation of and interaction with the UDF are nearly identical to the ones in the main program body seen in Figure 4.1.

As the current editor does not yet support information exchange between the editor and the system it is part of. A need for a way for the editor to take data into the element, such as program data or information about the available devices, needs to be added. The RIoT system will be used as an example and the main focus, as fully integrating it into the RIoT system is part of the thesis. The changes made and solutions discussed or displayed in relation to RIoT can be, with minor tweaks, applied to other smart environments, supporting scalability and reusability of the editor as well.

The program saving and user procedures logic is not implemented and will have to be designed before the implementation and integration. The new UDF support needs to introduce a way to assign RIoT device blocks to the given UDF in the program block in a way that will not affect the original UDF body. This will allow for the UDF to be used across the program as before, while abstracting the device selection and moving it from the procedure body to the place in the program where the UDF is used.

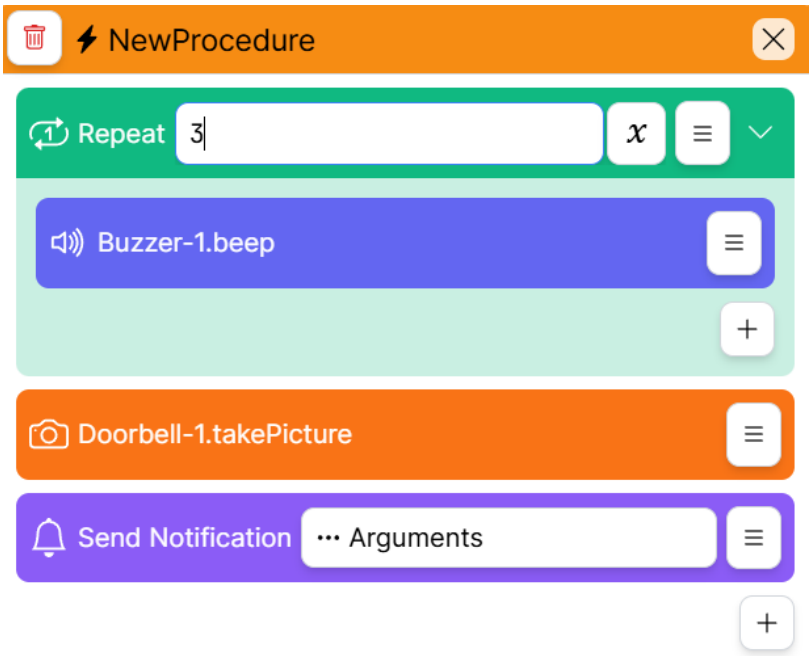

Figure 4.4: User procedure with simple program body opened for editing

The current state of the VPL Editor is well-suited for a new module integration that will introduce a solution more in line with the need for reusable UDF support while integrated into the RIoT system. As mentioned, two main aspects will have to be examined and have the solution tailored around them. The first one focuses on the VPL Editor and the representation, creation, and modification of UDFs within it, allowing for variable use of procedures while streamlining the creation process for the user. The second is the further integration of already existing functionality into the RIoT system, which dictates the need for the solution to support interactions with the database. Implementing CRUD operations for the program and user procedures, for which a need to be handled separately apart from the program arises.

The next section will examine existing solutions and framework advantages and drawbacks with respect to their relevance in regard to UDF creation and parametrisation, while not omitting the IoT integration. Relevant findings and conclusions for each existing solu-

tion will be outlined, with the final proposed solution reflecting them while keeping in line with the overall projects idea.

4.3 Existing Solutions

Most users are accustomed to interacting with intuitive and straightforward interfaces in daily-use applications such as messaging apps like WhatsApp and Messenger, or other useful or entertainment applications. These platforms rely on clear flows accommodated by simple gestures, clicking buttons or icons, dragging elements, or navigating through structured menus. Users should feel confident in navigating, creating, and modifying procedures without requiring prior technical knowledge or instructions.

In regard to the design and implementation, the user needs to have a clear way to create and utilise their own procedures. The creation of these procedures and their usage should be separated so as not to confuse the user due to possible nesting when creating more advanced and complex programs. The right approach must be chosen with respect to when the user will most likely define their own procedures and in what way they will be used. In the following subsections, two very popular, well-established solutions will be examined. Namely, Blockly's and Node-Red's approaches, as each framework offers unique insights into the two main aspects laid out in the previous section.

Blockly

Blockly provides a highly visual and intuitive approach to user-defined function creation, primarily through its drag-and-drop interface. Users define procedures by assembling blocks, each representing an action, input, or logical operation. To create custom functions, users can access the custom block creation tools, where they specify the block's structure, including inputs, outputs, and internal logic or may opt to define their custom blocks manually via JSON objects and JavaScript functions.

The more user-friendly approach is using the Blockly Developer Tools, seen in Figure 4.5. This visual tool utilises a similar block structure to allow the user to create their own custom blocks. The user is presented with the shell of the block, which defines its basic structure. Each block has a name, inputs defined via different pre-made blocks, block connections, a tooltip field, a help URL, and the colour of the block.

The pre-made blocks that are used to define custom blocks via the Blockly Developer Tools can be seen in Figure 4.6 from left to right for the Input, Fields, and Connection Checks. Using these, the user can create, set, and manage the input parameters of the block he is creating. In the example in Figure 4.5, a simple custom block is created using the value input block and the checkbox field type, with any subsequent connection after it. The connection type can be selected in the dropdown menu, allowing for only bottom connected, only top connected, top and bottom connected, or left connected type. Following these are the optional tooltip help URL variables with a colour picker at the end.

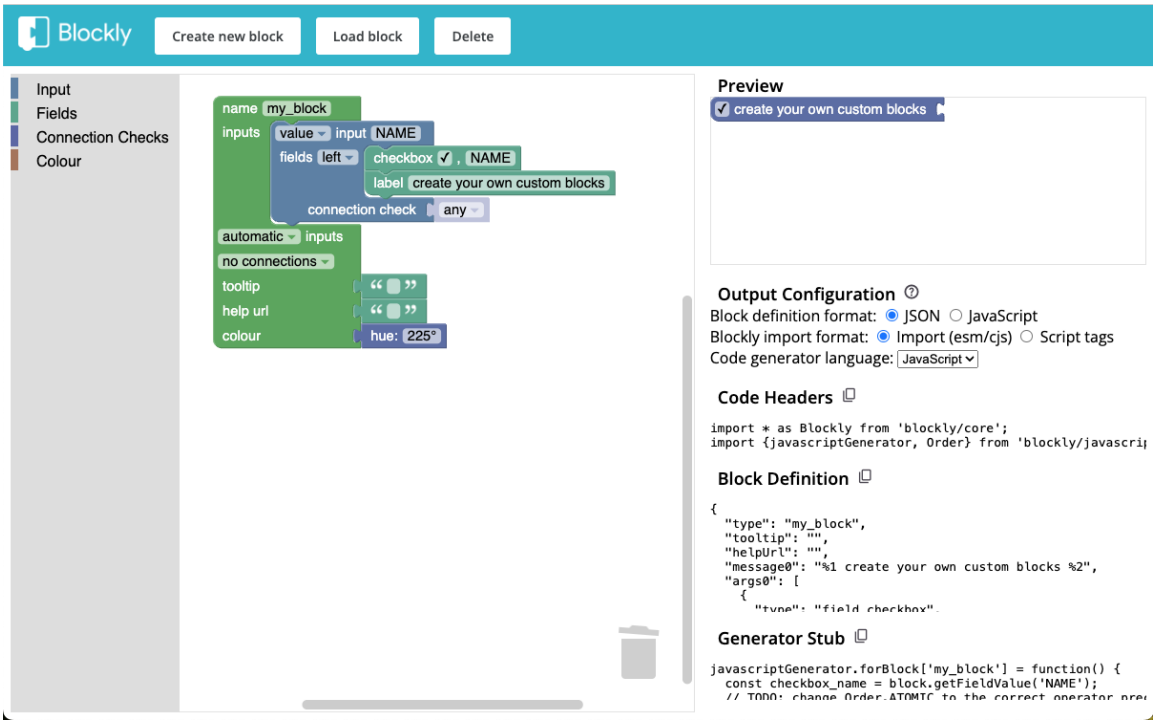

Figure 4.5: Block Creation via Blockly Developer Tools

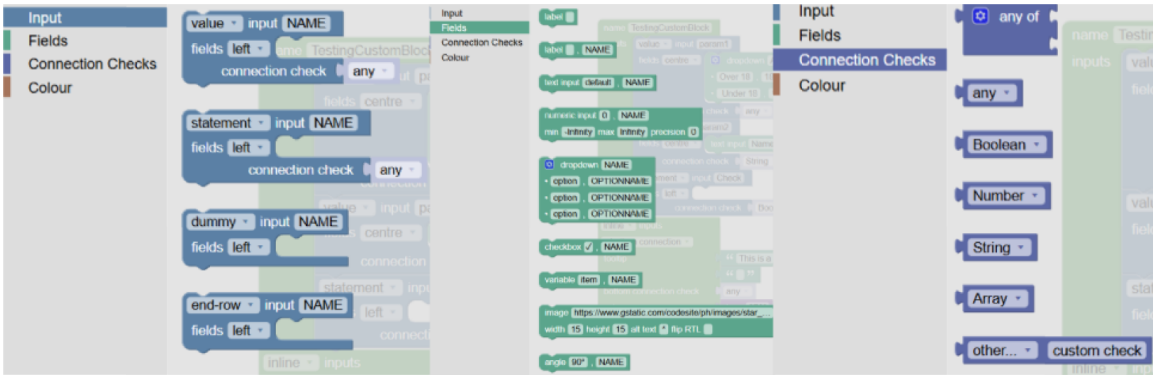

Figure 4.6: Options of Blocks in Blockly Developer Tools

All these come together to create a new custom block, which is visually displayed in the top right corner with the actual generated code for the block below it. To then use this custom block, the generated JSON code (Figure 4.7) for the block needs to be saved and included in the Blockly program in the correct order according to dependencies.

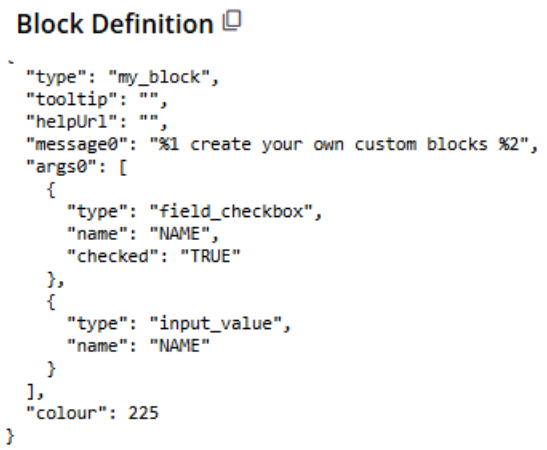

Figure 4.7: Code representation of Example Custom Block

For users who need to accommodate more complex concepts within the custom block, which might be difficult to create via the graphic interaction in the Blockly Developer Tools, a direct text-based block definition is available. The custom block is represented in the JSON format, seen in fFgure 4.7, letting the user precisely define the block to their needs. This can be accompanied by incorporating JavaScript into the block, allowing for far greater functionality and customisation. This requires more extensive programming knowledge and is not well-suited for the target audience, for whom the proposed solution is designed. On top of that, the main focus is on modification of the graphical editor within the VPL Editor, as opposed to the text editor, whose functionality will only be preserved.

Once defined, these blocks act as encapsulated units of functionality that can be visually incorporated into workflows by dragging them onto the canvas. This modular approach enables users to break down complex tasks into smaller, reusable components, enhancing clarity and efficiency. Blockly ensures that only valid block connections are permitted, guiding users through the procedure creation process and preventing syntax-related errors. This visual mechanism allows users to focus on creating the program logic rather than implementation details, making it accessible to beginners and non-technical users.

Although Blockly is effective as a visual programming tool, it faces limitations with complex or large-scale workflows, such as those occurring in IoT and smart environments. Highly nested logic or many interconnected blocks can result in visual clutter, complicating workflow management. Furthermore, Blockly lacks native support for Iot-specific features like device integration or communication protocols, often requiring additional custom implementation for such use cases. Custom blocks can be defined to represent the Iot devices and then used in the program in accordance with the general program's logic.

Blockly is conceptually similar to the existing implementation, outlining the need to keep the implementation and interaction logic in line with the general program's workflow when dealing with IoT devices or UDFs. Despite this, the way of defining reusable blocks which contain the IoT device does not accommodate use cases where the needed implementation logic is the same across multiple instances, but for different Iot devices. Furthermore, these custom blocks serve more to extend and modify the repertoire of available blocks, each serving their specific given purpose, rather than acting as custom user procedures. Tackling this issue will greatly improve the reusability of the UDFs and overall ease of use of the program.

Node-RED

Node-RED takes a different approach to EUDF creation, focusing on dataflow programming and seamless integration with IoT devices and services. Users define procedures by interacting with a flow-based editor, where each function or operation is represented as a node on a canvas. To define a custom procedure, the user selects the required nodes from a palette, drags them onto the canvas, and connects them using virtual wires to establish the data flow. Nodes can represent a variety of functions, including data input from sensors, computational processes, or control commands for devices. Users can configure each node by double-clicking it to open a settings panel, where they define parameters such as input formats, thresholds, or output destinations. The flow is executed from left to right, visually demonstrating how data moves through the procedure in real time, enabling users to understand and refine the process easily.

The creation of flows in Node-RED is rather easy. With some learning curve, even a user with minimal programming knowledge can create routines of varying complexity by using the provided nodes. The custom Node creation, however, is not so straightforward and user-friendly. Nodes get created when the flow is deployed [32], allowing them to send and receive messages while the flow is running and get deleted once the next flow is deployed.

Each Node consists of a JavaScript file that defines the Node's behaviour and an HTML file that defines the Node's properties, the edit dialogue and help text. The **package.json** is used to package them all together and as an npm module. To then include this Node into our current workspace, it needs to be installed via the npm install command. After successful installation, the node can be included in the code and interacted with as all the other nodes.

One of Node-RED's key strengths is its focus on IoT integration. As mentioned, the platform supports a wide range of protocols, such as MQTT, CoAP, and HTTP, and allows users to interact directly with IoT devices and cloud services. This makes Node-RED particularly well-suited for developing custom automation workflows in smart environments. Users can create highly tailored solutions by integrating multiple devices, services, and data streams into their procedures without major complications often associated with importing and connecting them to the VPL or other frameworks. This greatly reduces the need for custom Nodes as there is a wide variety of Nodes to be picked from, which slightly sets off the issue of custom Node creation difficulty.

This flexibility comes with a tradeoff. Node-RED's interface, while powerful, can be overwhelming for non-technical users. The process of configuring nodes and managing connections between them requires a level of understanding of dataflow programming and IoT architectures. This can make the learning curve steeper for users who lack technical expertise. Furthermore, Node-RED's workflows can become complex and difficult to manage as the number of nodes and connections increases.

The main takeaways from the Node-RED implementation give insight into the handling of the IoT device interactions and integration within the greater IoT system. While letting the user create and use custom reusable blocks, their behaviour and implementation in the workflow can be adjusted to better suit the situation they are used in.

4.4 RIoT

Lastly, the RIoT [9] needs to be examined and analysed to determine and outline the core aspects relevant to UDF support and VPL Editor integration. RIoT is a modular and

scalable information system designed for the management of smart environments consisting of various types of IoT devices [22], and the data processing and visualisation in dashboards [33] based on it. The architecture is based on a microservice model, where individual services are implemented in Go and communicate asynchronously using Rabbitmq. This allows for a highly responsive and fault-tolerant system suitable for horizontal scaling. On the frontend, a modern web interface built with React and TypeScript interacts with the backend via a Graphql API, enabling flexible access to system functionality and data.

A key feature of RIoT is its ability to monitor smart devices and evaluate their data using user-defined criteria, specifically Key Performance Indicators (KPIs). The system transforms raw sensor data into meaningful insights, which can then be used for visualisation, decision-making, or automated responses. This data can be used for variable creation when constructing conditions in the VPL Editor.

VPL Editor is installed into the RIoT as an npm package, introducing the `VPL-Editor` element used to facilitate all of the editor's functionality and interaction. RIoT does not have direct access to this element or its contents, while having access to all the information gained from IoT devices added to the system. A need for a data exchange between these two "layers" presents itself.

To accommodate and integrate the already existing VPL Editor's functionality into RIoT, CRUD operations on programs made by the user need to be introduced. This will allow for persistent saving of programs in the RIoT database. Once programs from the editor can be saved, additional interactions such as loading programs into the editor, updating and deleting existing programs can be added. This will help integrate the standing functionality while laying the base for further additions.

The data exchange in this case needs to be a two-way street, allowing for the data to be both taken from the VPL Editor and inserted into the RIoT for processing and vice versa. The crucial data needed from the RIoT is all the available IoT devices, as the editor is using a demo device list that currently does not reflect the state of devices in RIoT. Thus, providing the editor with all the devices and their commands in a format-compliant manner is crucial and will greatly move the integration, solidifying the VPL Editor's place and purpose within the RIoT system.

By combining the RIoT system's real-time capabilities with the visual interface of the VPL Editor, users are given a powerful and accessible toolset for defining logic and automating behaviours in smart environments. The integration supports both technical users and non-programmers, offering an intuitive way to design, manage, and run logic across complex IoT infrastructures while benefiting from the robustness and performance of a modern backend system.

4.5 Summary of Anaysis

Both Blockly and Node-RED offer valuable insights into the creation of EUDFs, each with its own strengths and limitations. Blockly excels in simplicity and user-friendliness, making it accessible to users with little technical experience while being conceptually more in line with the current implementation. Its visual clarity and error-prevention mechanisms make it ideal for smaller workflows and educational contexts. However, its lack of built-in IoT functionalities and challenges with managing complex workflows limit its applicability to advanced IoT use cases.

Node-RED, on the other hand, provides unparalleled flexibility and integration capabilities for IoT and smart environments. Its support for a wide range of protocols and

seamless device integration make it a powerful tool for creating custom automation workflows. However, its complexity and steep learning curve can be barriers for non-technical users, and its mobile usability is limited compared to Blockly.

From the analysis of the current VPL Editor, RIoT, and existing solutions, the following requirements shaping the proposed solution arise. The addition, creation, and use of custom user procedures should not disturb the flow of the user's experience, nor should they go out of their way to do so. It needs to be in line with the user's know-how and what they might expect from the application they are using and its current state presented to them. Building on top of the current support for UDFs in the VPL Editor, the logic of defining UDFs as standalone reusable blocks that adhere to the program's overall logic and workflow is correct. These blocks act as small standalone programs that get executed in place of the found user procedure ID in the program body.

The UDFs do not take parameters, so the only data that can flow into them are the device statements and user-defined variables placed in the body of the procedure. This standalone block logic will be kept due to the constraints of the implementation limitations and the goals of the thesis. Despite this, some sort of UDF parameterisation needs to be implemented with a focus on IoT device selection. Parametrisation allowing for greater flexibility in the use of procedures in a specific place of the program, similarly to Node-RED's parameterisation and ability to tweak and fine-tune nodes, needs to be introduced.

The need for a new and swifter way of constructing these procedures, while ideally allowing the user to reuse already crafted program parts, is crucial in mediating a frustration-free experience that users will gladly come back to again. To meet the integration requirement within the RIoT system, the VPL Editor must rely on and work with up-to-date information about the devices and values from sensors coming from the RIoT system, while being able to store needed information and data.

Chapter 5

Proposed Solution

The solution aims to utilise already existing frameworks and functionality to re-use and build on top of the established project structure. The solution consists of the changes to the editor, RIoT interface to accommodate new functionality, and the backend integration into the RIoT. The front-end will consist of the VPL Editor and the changes needed to accommodate the aforementioned advanced UDF support, and the RIoT-side UI, which will help to mediate the data exchange between the two. The back-end part will examine changes required to facilitate new blocks, possibly added to the language, used in programs and procedures that will be saved in the database. The programs saved in the database will be used in the `interpret-unit`, where they are parsed and evaluated. The new UDF logic will have to be reflected here as well.

The first front-end area is the modification of the VPL Editor to introduce more comprehensive user procedure creation and management logic. The standing logic and functionality will be updated to better distinguish it from the use case in the program body, while integrating it into the existing editor in an unintrusive manner. This will be accomplished via the Skeletonize logic, providing a faster construction of new UDFs utilising the selection of already existing program parts paired with parameterisation of the user procedures with IoT devices.

The second front-end is further integration of the VPL Editor into the RIoT system to accommodate sourcing of the aforementioned IoT devices registered in the RIoT system, while simultaneously adding the CRUD database operations to allow for persistent saving of the programs and user procedures. On the RIoT system side, the user procedures will be saved and handled separately from the program data to accommodate dynamic loading of current UDFs into the VPL Editor, making all available UDFs accessible to all programs being created and used.

The back-end area will focus on the updates to the database and operations required to facilitate the separation of programs and procedures while allowing for global access to procedures. The update to interpret logic should be minor, and changes needed to be made to the file to keep the original logic can be easily done before parsing takes place. As changes to procedure structure and representation will be made, the parsing will have to accommodate this lightly modified format.

5.1 Proposed VPL Editor Changes

The editor's basic functionality will be expanded upon by the populated UDF creation mechanism referred to as Skeletonize mode, allowing for a swifter and more enjoyable user experience while laying the base for the utility proposed later in the solution. This mechanism will introduce a new editor mode in which, instead of editing the program, the user selects the desired blocks, tracking them for addition to the body of the newly to be created procedure.

A simple diagram of the abstracted process seen in Figure 5.1 showcases the selection of blocks, preparation of the procedure and execution of other possibly necessary tasks prior to UDF population, and the newly created procedure populated by the selected block. The left block in Figure 5.1 represents the current program body, consisting of blocks described in more detail below, from which blocks are selected for the new UDF to be built. The middle represents an abstraction of the UDF preparation, which prepares the body of the new UDF and performs any necessary actions, with the block on the right representing the newly created UDF consisting of the selected blocks, where some were modified or changed to adhere to the new logic.

In selection mode, clicking on the block adds it to the tracking used in the UDF body preparation. The blue frame is a parent block allowing for child blocks, thus introducing nested logic, presenting a couple of ways to handle it while offering possible quality of life improvements for the user. This would come in the form of automatically selecting all nested blocks of the targeted block while still letting the user deselect the unsuitable ones. It will be faster and less tedious as opposed to selecting each individual block, and is more in line with what the user might expect.

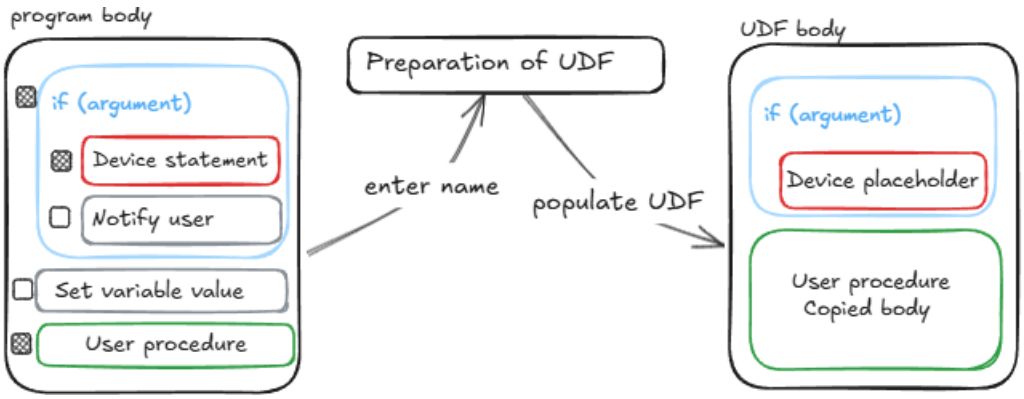

Figure 5.1: Block selection for the preparation of new UDF

The remaining colored frames each represent a different type of block, with grey representing the basic editor blocks, red being the device statement, and green symbolising a user procedure. This is to hint at the different behaviour of various selected blocks in the preparation of UDF. For example, the device statements selected will be replaced by a placeholder block, which holds block-type information instead of the actual device instance. This will abstract the device instantiation, allowing for modifiable assignment a the place of UDF usage in the program body instead of the original hard-coded devices. The user can create a procedure as before, but without having the option to add any device statements, with an option to use the placeholder block to use in building the procedure logic

in place of the original device statements. Device statements can still be inserted directly into the main program body, where the placeholder block cannot be inserted instead.

To initialise a procedure with the desired device parameters, the user clicks on the UDF block in the program body, opening the procedure initialisation modal where they can assign the chosen device to each individual placeholder. Instead of modifying the original procedure body, references to the selected devices will be made. When opening each initialised UDF in the program, its relative devices will be loaded, replacing their placeholders. This allows for great customizability while not modifying the original program.

The actual UI changes to the editor will include a slight `editor-controls` overhaul with a button for Skeletonize being added, visualisation of the selected blocks, and other visual element changes, tailoring a smooth experience. The initial proposed solution can be seen in Figure 5.2 with changes made to the graphical editor on the left and the body of the newly created UDF populated with the selected blocks, with devices being replaced by placeholders. The status of the editor will be visually conveyed to the user in an appropriate way. In the case of the draft, it is the red outline. The UI should prevent the user from making mistakes or introducing errors into the program while also guiding them via appropriate modification of the already familiar concepts.

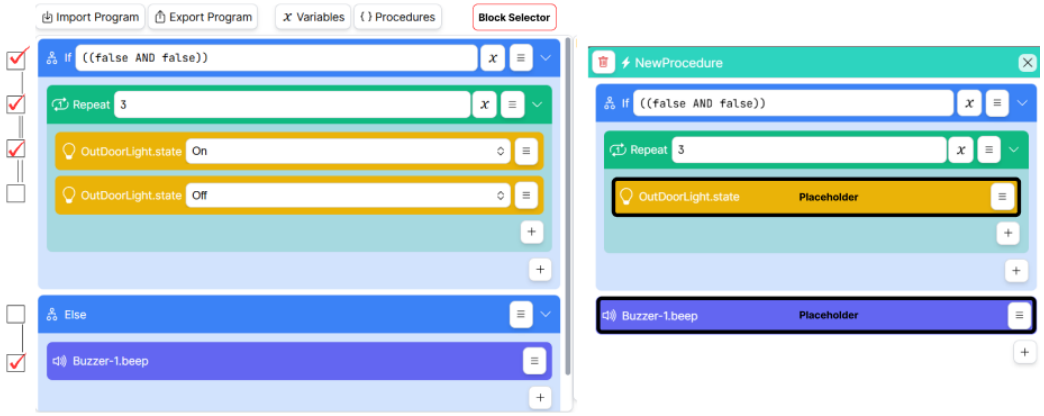

Figure 5.2: Proposed editor changes (left) and a placeholder logic (right)

5.2 Proposed RIoT Integration

At the moment, there is no interaction between the RIoT and the VPL Editor, with the editor being available for demo purposes. The basic overview of the hierarchy can be seen in Figure 5.3. The diagram displays the parts of the RIoT system with which the editor will communicate, or they will have to accommodate the changes made. Firstly, there needs to be an interface between RIoT and the VPL Editor to allow for data exchange.

The VPL Editor will be modified with a function accessible from the RIoT side, allowing for programs and devices to be passed to the editor. This will be used for the initialisation of the editor and during programs being loaded from the database. A way to keep track of all current procedures and their formatting suitable for the editor will be added.

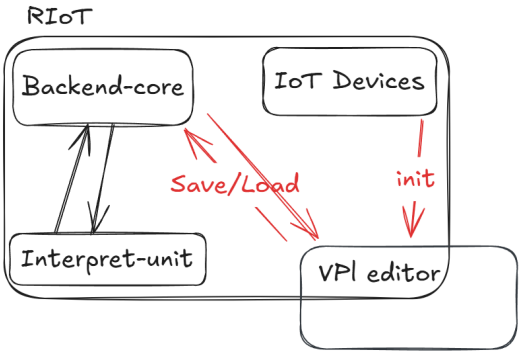

Figure 5.3: RIoT diagram

RIoT Front-end

When opening the editor from the Automations page, only the bare editor is displayed. The proposed solution will be modified and built upon to situate the RIoT interaction UI elements above the editor controls. These elements will provide all the necessary functionality for the implementation of database operations for programs and procedures, allowing for saving new ones, updating and deleting the existing ones, or updating just the changes made to procedures.

On the editor page load, when the editor component mounts, all the current devices with relevant information are fetched from RIoT and prepared into an editor-compliant format. The editor is relying on the consistency and correctness of device data received from the RIoT system and will not perform any additional checks.

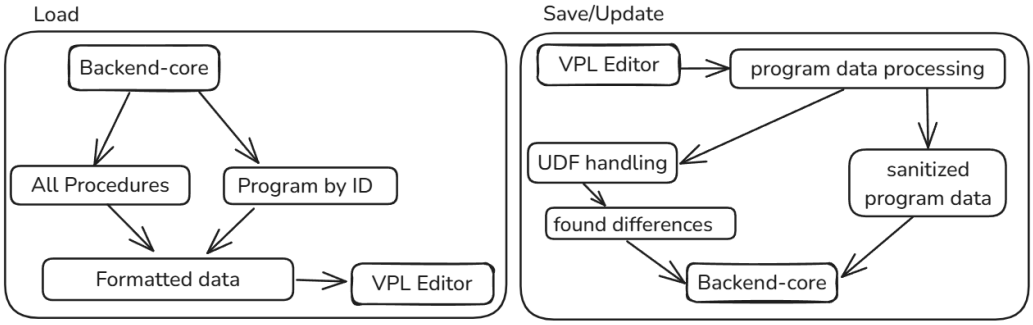

Figure 5.4: Flow chart showcasing the load and save/load process

RIoT passes the selected programs' data on the `Load` button click to the editor, where it is used to reinitialize the context and displayed to the user for use and interaction. Before the data is passed, all UDF entries are fetched, adding them to the header of the program, as seen in Figure 5.4, providing the user with all available procedures without having to save all procedures to each program saved into the RIoT database, preventing redundant and bulky program data.

On program save or update, the program data file is retrieved from the editor, saving the body and variables into the program database table. The user procedures from the program data are compared against the procedures table to determine changes that are then applied and reflected in the system. The simplified overview can be seen in Figure

5.4. This allows for modifying procedures while having the update made to it be applied to all programs using this procedure.

Back-end Integration

The back-end core will be modified to include the blocks added during implementation of proposed solutions, ensuring the validity check can be performed on program save to database and on program execute when it's sent to `interpre-unit` for parsing and execution.

A new table for VPL procedures will be created in the entry, holding its name and procedure body, allowing for dynamic loading without local restrictions. The program table will remain unchanged as the UDF separation takes place at the program data level by extraction of procedures from the header and subsequent clearance of the given collection. To future-proof the solution and provide a more solid base for functionality add-ons, a linking table between the program and procedures is proposed. Its functionality is fully fledged, but the general purpose will be to remove redundant data fetching and optimisation.

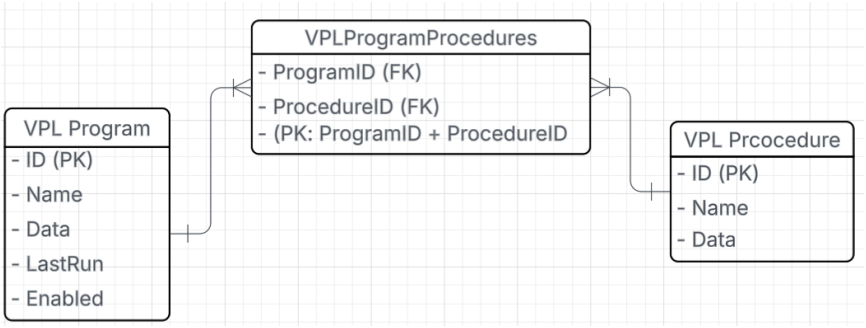

Figure 5.5: ERD of VPL Program and Procedure tables with an M:N relation

An entity relationship diagram, showcasing the solution described above, can be seen in Figure 5.5. It features three tables with two units that need persistent saving with an M:N relation implemented via a linking table. Linking table keeps a track of all unique procedures used in the program, allowing for better information propagation and ensuring data consistency.

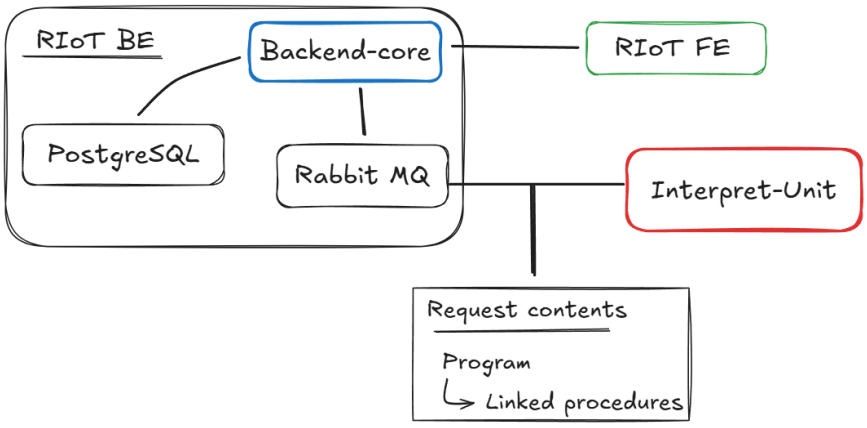

Figure 5.6: Request from Interpret-Unit to obtain the data for interpretation

This will also be utilised in the `interpret-unit`, ensuring the modifications done and new logic don't introduce errors or undefined behaviour like error parsing a missing UDFs, which will not be stored with the program anymore. While implementation details need to be discussed with respect to whether linking will be implemented, the general abstracted interaction of fetching the data from RIoT to interpret is described in a diagram seen in Figure 5.6. It visualises the relation and communication between the `backend-core` in the RIoT system and `interpret-unit` element. The `interpret-unit` queries the `backend-core` via `RabbitMQ` requests, fetching the program and user procedures for interpretation. Program Exec represents the entry point in `Interpret-Unit` seen in the previous Figure 5.6. It receives a program ID that was sent for execution, based on which all the necessary information is gathered, formatted, and prepared for interpretation.

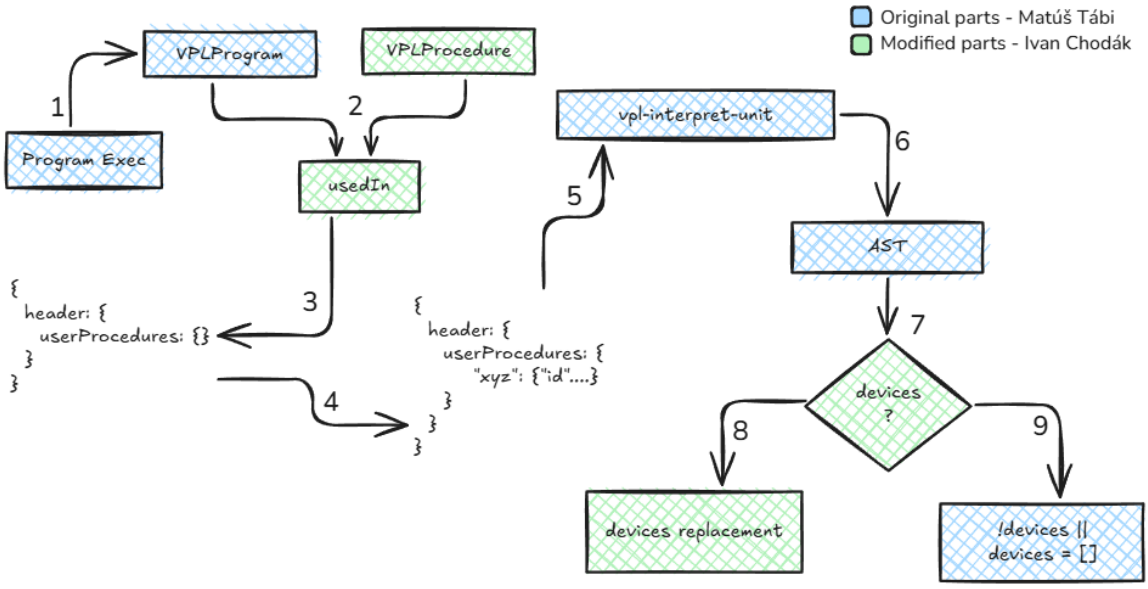

Figure 5.7: Diagram showcasing the interpret logic and proposed changes

The planned minor modification to the fetching and interpretation logic is visualised via the flowchart in Figure 5.7. The flowchart consists of blue statements, which are from the original implementation logic done by Matúš Tábi [28], with green ones highlighting the modified or added ones needed to accommodate reusable UDF support.

The program data for the interpretation is constructed from data of the given program, consisting of `userVariables` and the body block. This is fetched by the interpret query further using the program ID to fetch all the related procedures as seen in the 5.7 marked by the edge two. The empty `userProcedures`, formerly populated by all UDFs saved in each program, are populated by procedures fetched from the Backend-core database.

The query, consisting of either all available procedures or just the linked ones, is processed and parsed, inserting every entry into the program header seen in Figure 5.7 denoted by the edges three and four. After the program data is prepared and validated, it is passed to the `interpret-unit` for interpretation, marked by the edge five.

The only modification remaining to have the correct functionality restored is the device replacement in the initialized procedures. Placeholder blocks need to be detected before they are added to the AST, which is being created in the `interpret-unit` market by the edge six. During the parsing, user procedures are detected and checked for device place-

holder blocks. If no placeholders were found, denoted by the number nine, the parsing continues. On detection of the placeholder block, the corresponding entry from the procedure's metadata is retrieved, appending it instead of the placeholder block, denoted by the edge eight in Figure 5.7. This ensures that the proposed UDF custom initialisation and its implementation are correctly reflected in the final Go code.

Chapter 6

Implementation

The Chapter implementation of the VPL Editor follows the structure set out in the previous chapter 5, without any major changes or deviations from the original plan. An easier and more versatile way of UDF creation was provided via the Skeletonize functionality, controlled by an intuitive and visualized click-based selection of desired blocks. The UDF logic and structure were expanded to accommodate the data necessary for versatile user functions without hard-coded device statements.

The RIoT interface will be equipped with control elements to provide the user with a way to interact with the RIoT side of the integration. Taking care of the CRUD program operations and procedure updating, ensuring the changes from the VPL Editor are reflected and saved in the system. Device and procedure loading into the program will be done in the background, not burdening the user with its handling.

Minor changes will need to be made server-side to accommodate the new block, as files are validated on save. This change will need to be reflected in the interpret so that the device type blocks are replaced with the corresponding initialised device statement. This will ensure that the right selected logic is executed and appended to the parsing tree.

6.1 VPL Editor

As this thesis builds on top of an already existing implementation, reusing and modifying already existing assets while adhering to existing logic will be a priority to avoid inflating the project with unnecessary dependencies. The VPL Editor is built using the Lit library, utilising custom events to notify other parts of the editor of changes that might require a rerender of a given component. The proposed solution will not focus on the text editor, but its functionality needs to be intact after the new support for UDFS. The editor controls, various modals and information displayed or accessible from them, and primarily the graphical editor will be modified.

The program structure, as seen in Listing 4.2, consists of a header containing the user procedures and user variables entries, and a block which serves as the main body of the program. Currently, each header is unique to each given program, being together exported, imported, and handled as a single JSON file. The solution will propose a way allowing for the separation of user procedures, elevating them to be globally shared across all programs instead of the user procedures being locally locked to each program. The user variables will stay as local-only values to keep the custom variable interaction in the editor simple. The

structure will require a small revision and expansion to accommodate UDF parameterisation metadata.

While support for program saving and editing will be integrated, greatly increasing the reusability and bettering the user experience, it alone will not be sufficient to provide the level of customisation and re-usability required. The user might want to re-use only a smaller portion of the program or refactor its logic quickly, without the hassle of copying and editing the original program. This requires a simple yet effective way of letting the user select blocks from an existing program to populate the body of a newly created procedure, while not affecting the original program. This will be provided via the Skeletonize functionality button as seen in the modified editor controls in Figure 6.1.

Skeletonize

The Skeletonize mode will allow the user to select blocks from an existing program. The user turn the skeletonized mode on and off with the skeletonized button. The skeletonized mode will be accompanied by a short info element to familiarise the user with it. In the example seen in Figure 6.1, the first block was clicked, automatically selecting all its nested blocks. The set variable is not selected as the user clicked on it to deselect it from the skeletonized selection. Once the user picks the desired blocks, clicking on the `Create Procedure` button brings out the modal for procedure creation seen in Figure 4.3, where they enter the name and configure the visual aspects of the block. To streamline the process, the user should be taken straight to editing of the newly created procedure instead of having to click on it manually.

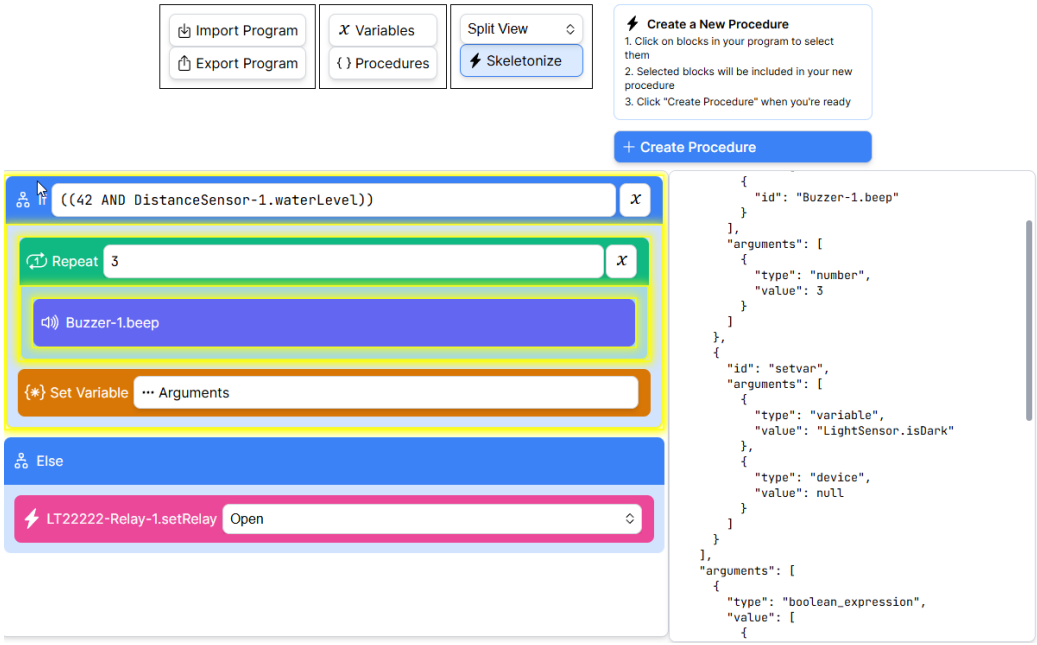

Figure 6.1: Example program used as source of the block selection for procedure population

The editing modal of the newly created procedure populated by the selected blocks can be seen in Figure 6.2. On the first glance, it is identical to the one in Figure 4.4, but there is a crucial difference. It is due to the lack of device statement blocks and the appearance of a new `deviceType` block, both of which will be discussed in greater detail

in the parameterisation section. The user can modify and finish up the creation of the new procedure, utilising all the basic blocks and selection of RIoT statements, in a similar manner to the main body.

Figure 6.2: Newly created UDF from Skeletonize and the available RIoT statements

The use of these procedures stays in line with the original implementation, where created procedure blocks appear in the `Add statement` selection among the basic blocks. In the next section, slight modifications will be introduced to further the UDF functionality while keeping it inline and ready with the RIoT integration. These changes focus on modifying the UDF representation logic to allow for parameterisation via assignment of devices from the RIoT system.

Parametrisation

As aforementioned, the UDF is contained in a standalone block which does not take any values as parameters, nor does it return any. Currently, parametrisation will not focus on or address this as it is in line with the current implementation logic. Parametrisation of UDFs is to be understood as abstracting the procedures by not allowing device instance blocks, which are replaced by a placeholder block in the procedure body. This separates the creation of UDFs and assignment of particular device instances, allowing for one logic to be reused in different scenarios or environments as opposed to being limited to hardcoded devices.

This is mediated and implemented by the Device Type block seen in Figure 6.2. It is an internal block, available only in the UDF editor modal, which holds the type and position obtained from the selected skeletonised device block. Optionally, the deviceType block can be directly added to the UDF without a type. The DeviceType block is not interpreted by the server as it doesn't execute any action.

This abstraction serves to provide a way to retain relevant information from the selected device statement block, but let the user assign any desired device block to the specific instance of UDF in the program body, as can be seen in Figure 6.3. The device statement blocks will now be available only in the main program block, thus allowing the user to create a simple program with directly initialised IoT devices without accidentally adding the device block to UDF and breaking the implementation logic.

The user procedure block will be extended by the `devices` array, holding references to the specific device type blocks in the procedure body. Each entry has uuid, which is used to link the device list entry to the correct device type in the UDF, a deviceId holding the

device type if uninitialized and the selected device statement if initialised, and the values array holding potential selected device statement values. This change, seen in Figure 6.3, will be accomplished by expanding the procedure block in the program body, originally consisting of only ID, with devices array made up of entries with uuid used for internal state management, the deviceID responsible for holding the name of the assigned device block, and values array holding the value for statements that have it.

Figure 6.3: Uninitialized (top) and initialised (bottom) UDF in the program body

By clicking on the added UDF in the program's body, such as the uninitialized procedure seen in Figure 6.3, the modal for procedure init is displayed with the body of the procedure loaded. On first procedure init, the body of the procedure is displayed, including any placeholder blocks in it. When the user clicks on any of the device placeholder blocks, a `Select Device Statement` modal is shown, displaying all the available device blocks. If the device type contains a valid type value, the displayed blocks are separated into two lists, with the top list showing devices of the corresponding type and the bottom showing all other devices. If the device type doesn't contain a valid type, only one list with all blocks is shown.

Upon device selection, the originally clicked device type block is replaced by the selected device statement. This replacement will not be directly modifying the body of the procedure as it would be with the original implementation. Instead, the selected device statement is saved into the corresponding entry in the devices array while keeping any of the potential values synced and recorded. The user will be able to reselect a different device statement by clicking on the target, bringing up the same modal for device statement selection, mirroring the device type use case. Clicking on initialised UDF will open the procedure init modal again, displaying the selected device statements in place of their corresponding device type block, as can be seen in Figure 6.4.

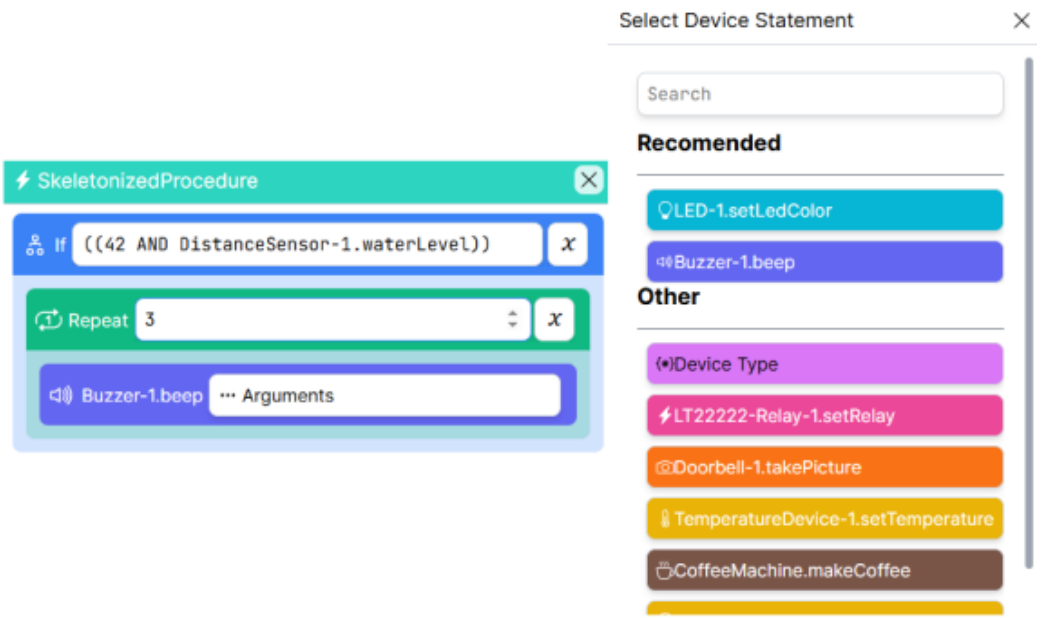

Figure 6.4: UDF opened for initialisation (left) and the device selection modal (right)

6.2 Integration into the RIoT System

The VPL Editor is currently featured as a standalone module available in the RIoT, accessible through the Automations page, seen in Figure 6.5, providing an interactive demo showcasing creation of programs in the Pocketix language. Despite this, there is currently no real integration between the RIoT and VPL Editor. The page hosting the VPL Editor in RIoT will be modified to accommodate necessary interactions, allowing for the integration, such as CRUD operations for the program and user procedures providing persistent storage or device loading, to ensure the VPL Editor is working with the latest devices in the RIoT system.

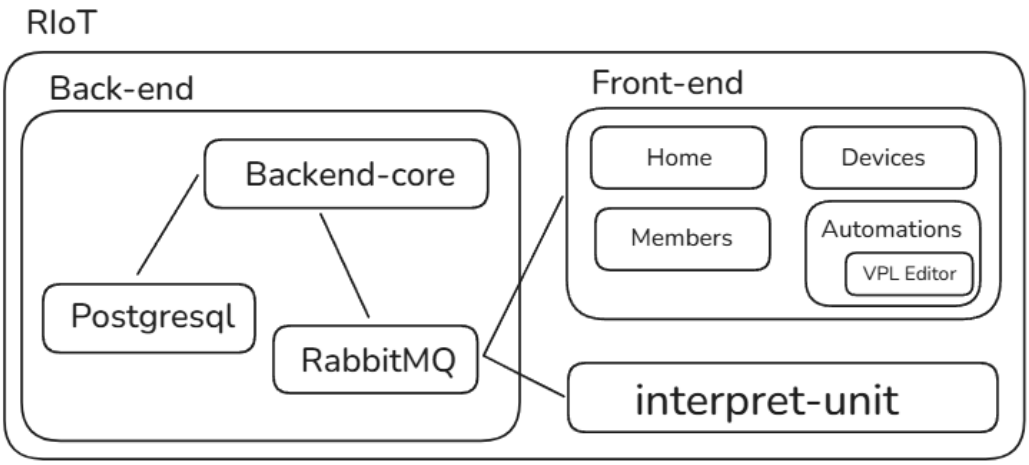

Figure 6.5: Overview of RIoT system modules

The user interface, seen in Figure 6.6, will feature a drop-down selector displaying all saved programs from the RIoT database. Clicking on a program from the list will select it, but no action will be performed yet. Three buttons under the program selector will serve as controls each providing the listed functionality of either `Load`, fetching the program data and passing it to the VPL Editor, `Update`, which takes the program data from VPL Editor and updates the selected program with it, and the `Delete`, which deletes the selected program. The second prominent UI feature will be a program name entry field with two buttons under it: the `Save Program` and `Update Procedures`. `Save Program` takes the name from the entry field and program data from the VPL Editor, using them to create a new program entry in the database. `Update Procedures` updates only the user procedures. This is for cases when only UDFs are modified.

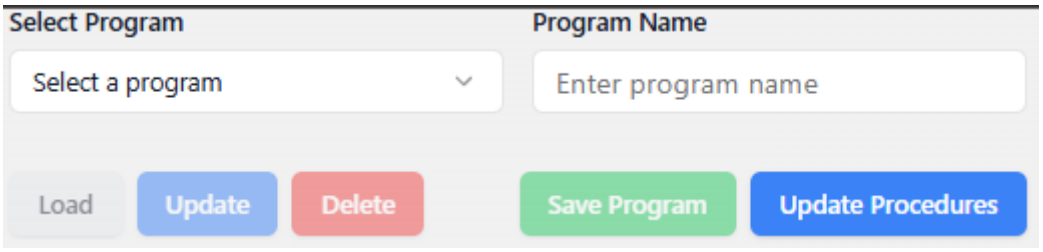

Figure 6.6: RIoT controls with program selector and name entry field

Database Integration

When the page is loaded and the VPL Editor component mounts, the device instances from RIoT are queried and formatted, preparing them to be passed to the VPL Editor. Similarly, all the user procedure entries from the database in RIoT are fetched, and `userProcedures` in the header of the program file is constructed. This data is then passed to the VPL Editor, initialising the editor with information corresponding to the current state of RIoT. Once this is finished, the editor is ready to be used.

Changes made in the VPL Editor will not be automatically reflected in the RIoT system, but due to executive decisions, will be operated by the above-described button-based UI from Figure 6.6. Once the user constructs a program in the editor, optionally containing new or modified UDFs, the program can be saved under a new name or be used to update the selected program. A simple overview of this process is seen on the right side in Figure 5.4, showcasing the processing of the UDFs from the program data before applying changes to the database. On `Save Program` button click, a non-empty name will be taken from the name input field, pairing it with the program data and `userVariables` pulled from the VPL Editor, creating a new database entry in the programs table. Subsequently, all necessary checks and validations related to program saving and procedure updating take place.

The device type block will need to be added on the server side as a valid block option, as validity checks will be performed server-side before a program or procedure is saved. One of these checks will include the status of UDFs in the program, looking for added, removed, or modified UDFs. These changes need to be detected and reflected in their respective entries in the database. This check is performed and applied anytime a `Save Program`, `Load`, `Update`, or `Update Procedures` button is clicked.

On any program data loading into the VPL Editor, the current state of the user procedures is constructed from the database, populating the `userProcedures` in the program

header before appending it to the rest of the passed data, as seen in Figure 5.4. This ensures that the VPL Editor is working with the latest versions of procedures, providing independent and complete access to every program.

A linking table, seen previously in Figure 5.5, between the programs and procedures used in them was implemented to provide more information and a better base for further implementation. This will allow us to fetch only the relevant procedure information needed for the interpretation of the chosen program, removing the redundant information that would be otherwise loaded.

Server-side Integration

The user procedure evaluation logic had to be adjusted to account for the new block type, updating the parsing function to replace the `deviceType` block with the corresponding device entry from the `devices` array. The separation of procedures from the program means that the retrieved program for interpretation does not contain user procedures in the header, unless this is a legacy program created before this thesis implementation. A need for a scalable and adequate solution to provide the interpret with procedures relevant to the parsed program arises, which was not originally accounted for in the proposed solution.

The agreed-upon solution was to implement a linking table of programs and used procedures, allowing for just the used procedures to be fetched on program interpretation. This was deemed as a more efficient and long-term resilient solution as opposed to bulk loading all the procedures or ad-hoc querying on an unknown block ID.

This table is queried with the ID of the program to be interpreted, returning linked procedures and their data. The combination of the procedure ID and its data is used to prepare a suitably formatted user procedure entry. The output is then parsed by the already implemented user procedure parsing function, adding it to the `header.userProcedures` of the program, which is extended by these entries.

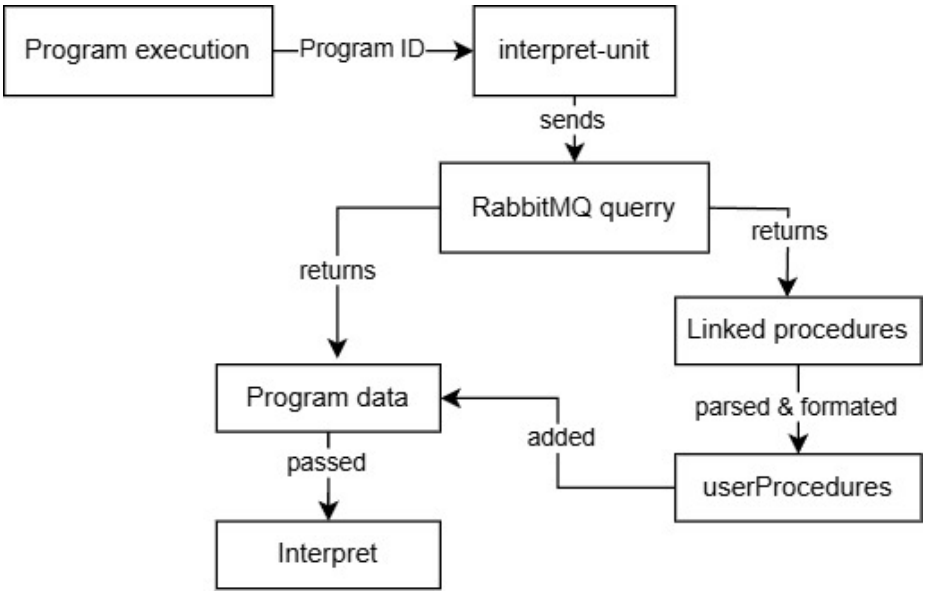

Figure 6.7: UML Diagram showcasing the fetching and formatting of UDFs in interpret-unit

Chapter 7

Testing

The main goal of the user testing is to evaluate the proposed and implemented solution in regard to the thesis goals, with the main aim to test the intuitiveness and level of quality of life improvement provided. Then overall feedback is reflected upon and used for the proposed future additions and improvements. The users will have the RIoT system introduced, followed by a demo showcase paired with a task for the user to try and execute, while being observed and asked about the current interaction experience. At the end, they express their overall feedback and suggestions regarding the tested experience. This feedback was then gathered and evaluated, drawing a concrete conclusion which was then formatted and reflected upon in future extensions.

7.1 User Testing

The RIoT system, its basic structure, and purpose are presented to the user to give them a general overview. The user is informed about the purpose, scope, and goals of the thesis, sparing them the technical details. The testing scenario will first ask the user to create a simple program, testing the program creation in order to subsequently test the program saving, loading, updating, and deletion. Then the user is tasked with creating a new UDF via the original procedure creation functionality to familiarize them with UDFs and their use in the program, while setting a baseline for comparison of the new functionality.

Following up on this, the new Skeletonize functionality and UDF initialisation are tested. The program created by the user is used in the skeletonize selection, where the task is to select blocks to transfer existing logic into a new UDF. This UDF is then edited and finished to be then used and initialized in the program. Lastly, the use of the new UDF is tested by the user, where they initialize it with the desired devices.

Each test case will follow the above-mentioned testing scenario, tested by a different target user with slight differences in the amount of information provided to simulate the real use case as best as possible. The testing will be mainly evaluated based on the following criteria. Intuitiveness, evaluating the ease of use and technologies or methods used in regard to users' overall expectations, to create a frustration-free experience they would return to. Quality of life aims to find out if the solution meets the thesis criteria to provide users with means to control a smart environment containing different IoT devices from one place, as opposed to using the native applications that come with each device. The feedback will capture the opinions and feelings of user towards the tested application and their remarks and suggestions, serving as indicators used for proposing relevant future additions.

Unassisted Design Intuitivness Testing

This case aims to gauge the difficulty of adapting the solution by a technically well-versed user, a male STEM University student with good computer knowledge, accustomed to many types of interactions with applications. Only the RIoT system and task at hand were explained, without instructions on how to use the editor. Aspects and functionality not important or relevant for testing are outlined to the user.

The user took time exploring different features and means of interaction, inquiring about some specific functionality or purpose of buttons and blocks, but overall could navigate the editor quite swiftly without major issues. Then the user was shown the original user procedure creation process with an explanation about device blocks and the logic necessary to use them. Skeletonize mode, with only its purpose described, was introduced, observing the subject attempt to complete the task of creating a simple procedure reusing an already existing program body. The user could complete this task intuitively without major issues.

The following task of using and initialising the newly defined procedure was a bit less straightforward. The user intuitively clicks on the used procedure to interact with it, but the opened modal for initialisation was too similar to the one for editing, confusing the user, making him close the modal and look elsewhere. After informing the user they were in the right section, they continued to attempt to initialise the procedure. The user struggled with the concept of initialisation of procedures with devices and asked more about it, with the explanation helping them successfully complete the task.

Overall, the user found the editor to be intuitive to use and in line with his expectations of interactions. The new selection logic was welcomed, and its quality of life improvement potential was recognised in potential real use by the user. The main drawbacks for the user were similarly looking elements serving different purposes without labels, informative descriptions or highlighted distinctions. Having the initialisation be entered through the statement block menu rather than block click would be more in line with the user's personal preference.

Target User Testing

The second test case is a target audience user testing involving a middle-aged female accountant, with average surface technical knowledge, living in a house with gradual introduction to smart devices and more advanced mobile functionality. This will simulate a real-life scenario where I figure as the RIoT system admin introducing a new user to the environment and showing them the functionality. It aims to gauge the accessibility of the solution to the user within the broader context of the system, while inquiring about their interest in such a solution within a smart-home environment.

The RIoT system, its modules, and purpose will be presented to give the user an idea for motivation of the thesis, with the goal of equipping the user with the ability to take control of this smart environment with ease. Following it was a brief presentation of the editor functionality and UDF creation with comparison to the original solution. User was then tasked to create their own program and perform CRUD operations on it, creating their own UDF via the selection logic, using and initialising it in the program.

While successfully completing all the tasks and giving positive feedback, the user required more assistance with searching for some of the desired functionality and was unsure in their initial use of the application. After some time and additional demonstration, the user picked up the flow of the application and could complete the task with more ease. The initial confusion was caused by similarly looking pages with slight differences between them,

paired with not having a current window name description, with potential instructions or help, while the yet unpolished design did not positively influence it.

Despite the initial hiccups, the tested solution was warmly received and acclaimed as an intuitive solution that the user would come back to, and can see themselves using in smart home management. The need for consistent and clear naming, with a visually telling UI overhaul featuring well-known and widely used features, was brought up by the user and is very important to them. They can imagine the current selection for procedure creation work well on a mobile platform, allowing for quick creation and use of simple programs, but the solution needs to be adjusted to suit complex programs better.

Genral feedback

Lastly, a demo was presented to a group of six bachelor students from the Faculty of Information Technologies from Brno University of Technology, accompanied by a master's student in the same field, and the thesis supervisor, Ing. Pert John. The thesis and its implementation were presented to receive both wider feedback for the implemented logic and suggestions for feature additions. The demo followed the established pattern of CRUD operation showcase, UDF creation via both the original and new selection logic, and UDF initialisation with the device statement instances as parameters.

The selection functionality was deemed an improvement to the original solution, providing the user with a swifter way of procedure creation. Need for a better, more user-friendly selection functionality button name was brought up, along with a better structure and display layout for the statement addition modal, having the procedures be displayed in their own tab. Having the database operations be implemented with a need for the user to click on fewer blocks to achieve their goal, while providing a better visual preview of the function before they select a UDF, was suggested among future extensions. Additionally, a new way for storing and previewing UDFs before they are used was proposed to further ease the use of procedures in the program body.

7.2 Future Extensions

Based on the results of the user testing and general feedback, several directions for future extensions are proposed to improve usability, clarity, and functionality of the editor. The most prominent issue observed across test cases was the visual and functional similarity of modals used for editing and initialization. Future work should include a redesign of these interfaces with clearer visual cues, labels, and purpose-distinctive layouts to eliminate user confusion and streamline task completion.

To further enhance the accessibility and ease of use, especially for non-technical users, the interface should include dynamic tooltips, contextual help, and clearly labelled buttons. These features would help users understand the available actions without requiring prior instruction. Additional improvements include better window naming and navigation aids, such as breadcrumbs or persistent headers, to provide orientation within the editor. A core change needed in naming is the confusion between programs and procedures. The similarly sounding names without a clear distinction cause confusion for the user, and procedures should be renamed to functions, as this is now more in line with their use and functionality.

A procedure store was proposed, displaying all available procedures and additional information about them, allowing the user to have an overview of their choices in procedures. This would remove the need to inspect the procedures by opening them or using them in the

body first, removing unnecessary steps needed to get the information the user needs. This would further allow for the sharing of procedures among different environments and systems. Need for value preservation when selecting the same device statement as was originally replaced by the placeholder block to streamline the use or a way to create predefined reusable UDF blocks was highlighted.

Another key extension is the implementation of a guided or tutorial mode for first-time users. This could walk them through the core functionalities, particularly the more abstract concepts such as UDF initialization with devices, which caused hesitation even among technically proficient users.

Given the positive reception of the Skeletonize-based procedure creation, it is advisable to further refine and expand this mechanism to support more complex pattern selections and transformation scenarios. Finally, adapting the interface for mobile platforms with an emphasis on usability for short and simple program creation would align with user expectations and increase practical adoption in smart-home environments.

Chapter 8

Conclusion

With the rising popularity of smart devices, there is a growing need for a simple yet powerful tool for home environments. The variety of IoT devices and the lack of unified control present a challenge. This is addressed by the RIoT system through gradual development and expanding functionality. It encapsulates various devices under a single logic and interface, providing users with advanced dashboard visualisations and a VPL Editor for creating programs and procedures that control and interact with the smart environment.

The thesis provides a comprehensive overview needed for a basic understanding of smart environments and their role in the creation of smart cities. Architectures and technologies are examined to gain a better perspective on the building blocks used in smart environments. The various usages of IoT are presented and their potential link to Smart Cities.

End-user development in visual programming languages is examined to support the thesis goals. Based on this, the existing solution is analysed, leading to a new, user-friendly procedure creation approach with reusability. The solution is implemented, tested with different users, and future extensions are proposed based on the results.

This thesis further integrated the editor into the RIoT system, enabling persistent data storage and passing of information, such as current devices, between the system and the editor. Two key modifications were made: a faster, more user-friendly procedure creation method and device parameterisation for functions, allowing procedure reuse within program logic. Together, these enhancements significantly improve the editor's functionality and bring the RIoT system closer to practical home use.

While the proposed solution met the thesis goals and enhanced RIoT's functionality to better support users, further development is required. The focus on implementation over UI design led to some usability issues identified during testing. However, users were still able to complete tasks successfully and provided positive feedback, with UI improvements noted as a recommended enhancement.

To address usability challenges revealed during testing, future development should focus on clearer distinction between editor modals, improved clarity through tooltips and navigation aids, and renaming "procedures" to "functions" for better conceptual alignment. A centralized function store should support reuse and cross-environment sharing. Preserving parameter values when replacing blocks, supporting predefined UDFs, and adding a guided tutorial mode are key for novice users. Enhancing Skeletonize and adapting the editor for mobile use would further improve flexibility and accessibility.

Bibliography

[1] Al Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M. and Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*. Ieee, 2015, vol. 17, no. 4, p. 2347–2376.

[2] Albino, V.; Berardi, U. and Dangelico, R. M. Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of urban technology*. Taylor & Francis, 2015, vol. 22, no. 1, p. 3–21.

[3] Ashton, K. et al. That 'internet of things' thing. *RFID journal*. Hauppauge, New York, 2009, vol. 22, no. 7, p. 97–114.

[4] Atzori, L.; Iera, A. and Morabito, G. The internet of things: A survey. *Computer networks*. Elsevier, 2010, vol. 54, no. 15, p. 2787–2805.

[5] Blackstock, M. and Lea, R. Toward a distributed data flow platform for the web of things (distributed node-red). In: *Proceedings of the 5th International Workshop on Web of Things*. 2014, p. 34–39.

[6] Bonomi, F.; Milito, R.; Zhu, J. and Addepalli, S. Fog computing and its role in the internet of things. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, p. 13–16.

[7] Bormann, C.; Castellani, A. P. and Shelby, Z. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*. IEEE, 2012, vol. 16, no. 2, p. 62–67.

[8] Boulogeorgos, A.-A. A.; Diamantoulakis, P. D. and Karagiannidis, G. K. Low power wide area networks (lpwans) for internet of things (iot) applications: Research challenges and future trends. *ArXiv preprint arXiv:1611.07449*, 2016.

[9] Bureš, B. M. *Systém pro zpracování dat z chytrých zařízení* online. Božetěchova 1/2, 612 66, Brno, 2024. Bachelors Thesis. FIT VUT Brno. ISBN 153937. Supervisor Hynek Jiří, P. Available at: `https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=273709`. [cit. 2025-5-10].

[10] Caragliu, A.; Del Bo, C. and Nijkamp, P. Smart cities in Europe. *Journal of urban technology*. Taylor & Francis, 2011, vol. 18, no. 2, p. 65–82.

[11] Carnegie Mellon University, S. o. C. S. *The Only Coke Machine on the Internet* online. 1998. Available at: `https://www.cs.cmu.edu/~coke/history_long.txt`. [cit. 2024-11-9].

[12] Cocchia, A. Smart and digital city: A systematic literature review. *Smart city: How to create public and economic value with high technology in urban space.* Springer, 2014, p. 13–43.

[13] Curtis, N. *Modular web design: creating reusable components for user experience design and documentation.* New Riders, 2010.

[14] Fraser, N. Ten things we've learned from Blockly. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond).* 2015, p. 49–50.

[15] Gan, G.; Lu, Z. and Jiang, J. Internet of Things Security Analysis. In: *2011 International Conference on Internet Technology and Applications.* 2011, p. 1–4.

[16] Gan, G.; Lu, Z. and Jiang, J. Internet of things security analysis. In: IEEE. *2011 international conference on internet technology and applications.* 2011, p. 1–4.

[17] Gubbi, J.; Buyya, R.; Marusic, S. and Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems.* Elsevier, 2013, vol. 29, no. 7, p. 1645–1660.

[18] Harrison, C. and Donnelly, I. A. A theory of smart cities. In: *Proceedings of the 55th Annual Meeting of the ISSS-2011, Hull, UK.* 2011.

[19] Hossain, M. M.; Fotouhi, M. and Hasan, R. Towards an analysis of security issues, challenges, and open problems in the internet of things. In: IEEE. *2015 ieee world congress on services.* 2015, p. 21–28.

[20] International Telecommunication Union (ITU). *Mobile phone ownership* `https://www.itu.int/itu-d/reports/statistics/2023/10/10/ff23-mobile-phone-ownership/`. 2023. [Accessed: 2024-11-10].

[21] John, I. P. *Prof. Ing. TOMÁŠ HRUŠKA, CSc.* Obtained on request. Božetěchova 1/2, 612 66, Brno, 2024. Dissertation Thesis. FIT VUT Brno. Supervisor Ing. TOMÁŠ HRUŠKA, C. prof. Available at: `Obtainedonrequest`. [cit. 2025-5-10].

[22] Joukl, M. *Web Application for Managing Smart Devices* [online]. Brno, 2025. Bachelor's Thesis. Brno University of Technology, Faculty of Information Technology. Available at: `https://www.vut.cz/en/students/final-thesis/detail/161070%7D`. [cit. 2025-05-02].

[23] Kelleher, C. and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR).* ACM New York, NY, USA, 2005, vol. 37, no. 2, p. 83–137.

[24] Khan, R.; Khan, S. U.; Zaheer, R. and Khan, S. Future internet: the internet of things architecture, possible applications and key challenges. In: IEEE. *2012 10th international conference on frontiers of information technology.* 2012, p. 257–260.

[25] Komninos, N. *Intelligent cities: innovation, knowledge systems and digital spaces.* Routledge, 2013.

[26] Komninos, N.; Bratsas, C.; Kakderi, C. and Tsarchopoulos, P. Smart city ontologies: Improving the effectiveness of smart city applications. *Journal of Smart Cities*, 2019, vol. 1, no. 1, p. 31–46.

[27] Light, R. A. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2017, vol. 2, no. 13, p. 265.

[28] Matúš, T. *Zpracování vizuálního programovacího jazyka na straně serveru.* Božetěchova 1/2, 612 66, Brno, 2025. Bachelors Thesis. FIT VUT Brno. Supervisor John, I. P.

[29] Nam, T. and Pardo, T. A. Conceptualizing smart city with dimensions of technology, people, and institutions. In: *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times.* 2011, p. 282–291.

[30] Nardi, B. A. *A small matter of programming: perspectives on end user computing.* MIT press, 1993.

[31] Navani, D.; Jain, S. and Nehra, M. S. The Internet of Things (IoT): A Study of Architectural Elements. In: *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS).* 2017, p. 473–478.

[32] Node-RED Developers. *Creating Nodes in Node-RED* `https://nodered.org/docs/creating-nodes/`. 2025. [Accessed: 2025-01-25].

[33] Olešák, M. *Responsive Visualization of IoT Data Using Dashboards* [online]. Brno, 2025. Bachelor's Thesis. Brno University of Technology, Faculty of Information Technology. Available at: `https://www.vut.cz/en/students/final-thesis/detail/161069%7D`.

[34] Pane, J. F. and Myers, B. A. *Usability issues in the design of novice programming systems.* Carnegie-Mellon University. Department of Computer Science, 1996.

[35] Perera, C.; Zaslavsky, A.; Christen, P. and Georgakopoulos, D. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials.* Ieee, 2013, vol. 16, no. 1, p. 414–454.

[36] Pew Research Center. *Mobile fact sheet* `https://www.pewresearch.org/internet/fact-sheet/mobile/`. 2023. [Accessed: 2024-11-10].

[37] Podvojský, B. L. *Vizuální programování IoT zařízení* online. Božetěchova 1/2, 612 66, Brno, 2024. Bachelors Thesis. FIT VUT Brno. ISBN 154356. Supervisor Hynek Jiří, P. Available at: `https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=265818`. [cit. 2025-5-10].

[38] Rashwan, O. A. Simulink as Multidisciplinary approach in Teaching Engineering Mathematics. *International Journal of Scientific & Engineering Research*, 2019, vol. 10, no. 13, p. 501–505.

[39] Resnick, M.; Maloney, J.; Monroy Hernández, A.; Rusk, N.; Eastmond, E. et al. Scratch: programming for all. *Communications of the ACM*. ACM New York, NY, USA, 2009, vol. 52, no. 11, p. 60–67.

[40] Sabella, R. R. *NFC for Dummies*. John Wiley & Sons, 2016.

[41] Saleem, Y.; Crespi, N.; Rehmani, M. H. and Copeland, R. Internet of things-aided smart grid: technologies, architectures, applications, prototypes, and future research directions. *Ieee Access*. IEEE, 2019, vol. 7, p. 62962–63003.

[42] Satyanarayanan, M. The emergence of edge computing. *Computer*. IEEE, 2017, vol. 50, no. 1, p. 30–39.

[43] Shi, W.; Cao, J.; Zhang, Q.; Li, Y. and Xu, L. Edge computing: Vision and challenges. *IEEE internet of things journal*. Ieee, 2016, vol. 3, no. 5, p. 637–646.

[44] Sutherland, I. E. Sketch pad a man-machine graphical communication system. In: *Proceedings of the SHARE design automation workshop*. 1964, p. 6–329.

[45] Thangavel, D.; Ma, X.; Valera, A.; Tan, H.-X. and Tan, C. K.-Y. Performance evaluation of MQTT and CoAP via a common middleware. In: IEEE. *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. 2014, p. 1–6.

[46] Ur, B.; McManus, E.; Pak Yong Ho, M. and Littman, M. L. Practical trigger-action programming in the smart home. In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2014, p. 803–812.

[47] Want, R. An introduction to RFID technology. *IEEE pervasive computing*. IEEE, 2006, vol. 5, no. 1, p. 25–33.

[48] Whitley, K. N. Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing*. Elsevier, 1997, vol. 8, no. 1, p. 109–142.

[49] Wu, M.; Lu, T.-J.; Ling, F.-Y.; Sun, J. and Du, H.-Y. Research on the architecture of Internet of Things. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. 2010, vol. 5, p. V5–484–V5–487.

[50] Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L. and Zorzi, M. Internet of things for smart cities. *IEEE Internet of Things journal*. Ieee, 2014, vol. 1, no. 1, p. 22–32.

[51] Zapier. *Zapier Workflows - Automate your tasks and connect your apps* `https://zapier.com/workflows`. 2024. [Accessed: 2024-12-29].

Appendix A

Contents of the external storage media

- This file in PDF format.
- source/README.md - detailed description of the repository and changes
- source/RIoT – changed files in the RIoT system
- source/vpl-for-things – changed files in the original editor
- source-latex – folder with source files for building latex document