Universidade Estadual de Londrina

RAFAEL FIGUEIRA GONCALVES

MDD4CPD: SOFTWARE DEVELOPMENT FRAMEWORK FOR CYBER-PHYSICAL DEVICES

LONDRINA

2023

RAFAEL FIGUEIRA GONCALVES

MDD4CPD: SOFTWARE DEVELOPMENT FRAMEWORK FOR CYBER-PHYSICAL DEVICES

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. André Luís Andrade Menolli

LONDRINA

2023

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Goncalves, Rafael Figueira.
MDD4CPD: Software Development Framework for Cyber-Physical Devices / Rafael Figueira Goncalves. - Londrina, 2023.
137 f. : il.

Orientador: André Luís Andrade Menolli.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2023.
Inclui bibliografia.

1. Internet of Things - Tese. 2. Code Generation - Tese. 3. Software Quality - Tese. I. Menolli, André Luís Andrade. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 519

RAFAEL FIGUEIRA GONCALVES

MDD4CPD: SOFTWARE DEVELOPMENT FRAMEWORK FOR CYBER-PHYSICAL DEVICES

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. André Luís Andrade Menolli
Universidade Estadual de Londrina

Prof. Dr. Adilson Luiz Bonifácio
Universidade Estadual de Londrina – UEL

Prof. Dr. Edson Alves de Oliveira Junior
Universidade Estadual de Maringá – UEM

Prof. Dr. Rodolfo Miranda de Barros
Universidade Estadual de Londrina – UEL

Londrina, 12 Abril de 2023.

GONCALVES, R. F.. **MDD4CPD: Software Development Framework for Cyber-Physical Devices**. 2023. 137f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

A Internet das Coisas oferece diversas possibilidades para soluções inovadoras usando dispositivos conectados. No entanto, a adoção dessa tecnologia também apresenta novos desafios, especialmente para desenvolvedores com conhecimento limitado em domínio específico para a Internet das Coisas, que podem ter dificuldade em criar soluções eficazes, deixando o software vulnerável a problemas como manutenção e segurança. Para contornar esses desafios, esta dissertação propõe um framework de desenvolvimento de software baseado em metodologia orientada a modelos (MDD) para apoiar a criação de novas soluções para a Internet das Coisas. As principais características do framework incluem um meta-modelo baseado em hardware, uma etapa unificada de modelagem, bem como um processo de geração de código instantâneo e completo. O framework proposto é concretizado por meio de uma ferramenta Low-Code que modela visualmente dispositivos ciberfísicos e gera código. O framework foi validado por meio de comparação por métricas de software, dentro do escopo limitado foi capaz de gerar código Arduino, com alta qualidade de software, e sem perda de desempenho. A validação externa foi aplicada por meio de um Modelo de Aceitação de Tecnologia, o que considerou a ferramenta fácil de usar e útil. A ferramenta de código aberto está disponível em https://github.com/rzfzr/mdd4.

Palavras-chave: Internet das Coisas. Geração de Código, Qualidade de Software.

GONCALVES, R. F.. **MDD4CPD: Software Development Framework for Cyber-Physical Devices**. 2023. 137p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2023.

ABSTRACT

The rapidly evolving concept of the Internet of Things offers endless possibilities for innovative solutions using connected devices. However, the adoption of this technology also presents new challenges, particularly for developers with limited domain knowledge who may struggle to create effective solutions, leaving software vulnerable to issues such as maintenance and security. To address these challenges, this dissertation proposes a software development framework based on model-driven methodology (MDD) to facilitate the creation of new solutions for the Internet of Things. The framework's key features include a hardware-based meta-model, a unified modeling step, as well as an instant and complete code generation process. The framework is complemented by a Low-Code tool that visually models cyber-physical devices and generates code. The framework was thoroughly validated, and it was able to generate Arduino code, with high software quality, without any performance loss. External validation deemed the tool to be easy to use and useful. The Open-Source tool is readily accessible at https://github.com/rzfzr/mdd4.

The Internet of Things offers various possibilities for innovative solutions using connected devices. However, the adoption of this technology also presents new challenges, especially for developers with limited knowledge in the specific domain of the Internet of Things, who may have difficulty in creating effective solutions, leaving the software vulnerable to issues such as maintenance and security. To overcome these challenges, this master's thesis proposes a software development framework based on model-driven methodology (MDD) to support the creation of new solutions for the Internet of Things. The main features of the framework include a hardware-based meta-model, a unified modeling step, as well as an instant and complete code generation process. The proposed framework is implemented through a Low-Code tool that visually models cyber-physical devices and generates code. The framework was validated through software metric comparisons and, within the limited scope, it was capable of generating Arduino code with high software quality and without performance loss. External validation was applied through a Technology Acceptance Model, which considered the tool easy to use and useful. The open-source tool is available at https://github.com/rzfzr/mdd4.

Keywords: Internet of Things. Code Generation. Software Quality.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ARM	Advanced RISC Machines
CAD	Computer Aided Design
CIM	Computation-Independent Model
CPD	Cyber-Physical Device
CPS	Cyber-Physical System
CYM	Cyber-Physical Model
DSL	Domain-Specific Language
GQM	Goal Question Metric
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
ISO	International Organization for Standardization
LED	Light Emitting Diode
MCU	Micro-Controller Unit
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
OMG	Object Management Group
OO	Object-Oriented
PIM	Platform-Independent Model
PSM	Platform-Specific Model
QoS	Quality of Service

RISC	Reduced Instruction Set Computer
TAM	Technology Acceptance Model
UML	Unified Modeling Language
URL	Uniform Resource Locator

CONTENTS

1 INTRODUCTION

The percentage of internet-connected individuals grew 1125% in 19 years [4], and according to Howell [5], the number of connected devices (not considering personal computers and smartphones) is going to reach 125 billion by 2030 (from 27 billion in 2017). Internet of Things (IoT) research and implementations have grown exponentially, with the objective of improving personal comfort with the use of an ever-growing number and types of connected devices.

With IoT's rise in popularity, we also observe a concurrent increase in the number of new developers. Authors such as Schwab & Davis [6] request for each individual person to take part in the 'humanization of things', stating that anyone is capable to develop IoT solutions. Gubbi et al. [7] also mention IoT as being part of the next evolution of technology, where applications are only limited by the creativity of their creators, not by the technologies themselves.

IoT development is challenging due to its novelty, which presents difficulties not found in conventional platforms such as mobile and desktop. One of the biggest obstacles is ensuring a required availability and security, despite the limited computational power and battery present on many IoT devices. This requires not only efficient solutions, but also a deep understanding of the technology and its limitations. As a result, new developers may find it challenging to create precise solutions for IoT [8].

Model-Driven Development (MDD) is a software development methodology that aims to facilitate software development. A key aspect of MDD is the shift of responsibility from developers to models, decreasing domain knowledge requirements while facilitating the adoption of standards [9].

MDD involves transforming models of different levels of abstraction into source code. Typically, MDD uses three levels of modeling: Computation-Independent Model (CIM), which may only include domain and business models; Platform-Independent Model (PIM), which Ayed, Delanote & Berbers [10] define as lacking platform-specific technical details and being able to model system functionality across multiple platforms; and Platform-Specific Models (PSM), which contain the necessary definitions for deployment. The three hierarchical model levels undergo a series of transformations, resulting in the generation of executable source code.

MDD approaches must introduce a domain-specific language (DSL), applied to specify constraints, transformation rules, and models. The majority of these solutions use textual modeling, usually, the DSL is at a higher abstraction level than the target source-code, reducing overall source-code quantity. Some approaches use graphical DSLs

for modeling, resulting in even less source-code.

MDD approaches present lowering source code quantity, called Low-Code, and it has been garnering progressively more attention in recent years. Low-Code offers a solution to the increased demand for software development without requiring expert coding skills. Low-Code is a methodology that enables developers to create applications with minimal coding efforts by using pre-built modules and templates. This approach allows for faster development and easier maintenance, as application changes can be made quickly and easily [11]. Sahay et al. [11] state that Low-Code has Model-Driven principles at its core, such as the reliance on automation, analysis, abstractions, modeling, and meta-modeling.

Low-Code Development platforms typically offer a visual interface for building applications, reducing the need for extensive coding knowledge. These platforms often feature drag-and-drop components and pre-built templates, which can be customized to meet specific requirements. The resulting applications are usually scalable and can be deployed across multiple platforms.

Low-Code Development is gaining popularity due to its ability to reduce the time and cost of software development, while still producing high-quality applications. This approach is particularly beneficial for organizations looking to improve their digital capabilities, as it allows for faster delivery of applications and greater flexibility in adapting to changing business needs [11].

The development of the Internet of Things is a multifaceted endeavor that demands proficiency in both hardware and software. To address this challenge, the use of well-defined models can greatly assist the development of effective IoT solutions. MDD and Low-Code are two powerful tools that can be used to achieve this as mentioned by Rafique et al. [12]. However, it is important to note that specific tools are required for MDD adoption. These tools are necessary to ensure that the models generated are accurate and effective. The proposed MDD4CPD framework is one such tool, designed to aid in the generation of high-quality models and code for cyber-physical devices. Quality analysis using the MDD4CPD framework is also available, providing valuable insights into the effectiveness of the generated models and code [13].

1.1 Motivation

IoT faces numerous challenges as it matures, these solutions are intrinsically complex and vulnerable when compared to better-established desktop or mobile paradigms, they have higher requirements for availability, speed, as well as other challenges [14]. For example, the low computational power in the node devices and the high vulnerability for security issues, are described by Alaba et al. [15]. The fundamental need for integrating with other services, and a previously unmatched concern with energy consumption only present more challenges to the systems.

The extensive selection of electronics modules and suppliers defines the intrinsic heterogeneous world where the IoT ecosystem is placed. When combined with the mentioned new technical challenges, IoT development becomes an even harder endeavor, especially for new developers and companies.

The higher level of complexity in IoT solutions directly impacts their quality, which is an important concern and a fundamental characteristic of IoT. Motta, Oliveira & Travassos [16] have found quality to be the most agreed-on IoT concern between the 3 groups pooled, Literature Review, Practitioners, and Governmental Reports. One reason for this higher complexity is the fact of a single solution requires multiple distinct segments to work. These segments can be divided into server-side, communication infrastructure, data treatment, etc. This work focuses on client-sided, the physical device development. This facet is defined by Atzori, Iera & Morabito [17] as "Things-oriented" vision, including the implementation of smart devices, and modules, excluding infrastructure or semantic matters. These devices are referred by Cyber-Physical Devices (CPD) in our work.

Ahmed et al. [18] state that due to market pressure, IoT solutions, particularly cyber-physical, have been released without sufficient testing, which demonstrates quality issues. The most popular issue with IoT is security, which has the greatest presence in the media, in particular with smart toys and the easily compromised recordings or data of its users.

Recon [19] presented 14 vulnerabilities in a smart-doll called "Hello Barbie", due to poor design choices and implementation, despite acknowledging the high amount of consideration the company had put into their software and hardware security. Some of the vulnerabilities were: unencrypted WiFi and communication, as well as no session expiration or password brute force protection. The vulnerabilities were present in all fronts, servers, and hardware, which allowed exploiters to easily steal passwords and voice recordings from users (children), and redirect these users to different websites.

MDD is a methodology that is particularly suitable for addressing the software quality challenges inherent in the development of IoT and cyber-physical devices, which is supported by multiple authors [2] [20] [21].

However, for MDD to be effective in developing IoT solutions, an appropriate tool specifically tailored for this purpose is required. Kapteijns et al. [22] pointed out that over a decade ago, the literature highlighted the lack of such a tool. The longevity of this recognized gap emphasizes the importance of addressing this issue in the field of IoT development. Currently tool unavailability is confirmed in the literature, for instance, a 2018 survey Akdur, Garousi & Demirörs [23] concluded that the lack of Tool support was the number one encountered challenge when applying MDE (with 75.70% votes out of the 627 practicing participants).

1.2 Objectives

The main objective of this work is to develop a low-code and MDD-based solution, to aid in the generation of quality cyber-physical solutions. The specific objectives of this work are:

1. Develop a software framework for cyber-physical devices.

   The framework will support a wide range of devices via specified models and transformations. The main contributions of the framework are a meta-model and a single modeling step.

2. Develop a prototype tool, for the proposed framework.

   The tool will provide a modeling canvas for the modeling specified by the framework, and apply the transformations into source-code.

3. Execute Internal Validations.

   Validations to judge the generated solutions' efficacy and software quality characteristics.

4. Execute External Validations.

   Validations to judge the framework's usefulness and ease of use.

2 BACKGROUND

This chapter presents the theoretical foundations for relevant topics. We describe the current state of the Internet of Things (IoT), its definitions, and the software quality paradigm. Then the chapter presents the main aspects of MDD. In the final section, we review the relevant existing research, providing a brief description of the Systematic Mapping used to obtain them.

2.1 Internet of Things

Asghar, Negi & Mohammadzadeh [24] define IoT as an extension of the internet in the real world, by connected objects. Atzori, Iera & Morabito [17] produced a survey exposing the technology's main challenges, their simplest definition involves the concept of universal presence, where an object must fulfill its purpose in an abstract manner.

IoT enables a wide array of applications, Gubbi et al. [7] argues that these solutions promote innovations, being limited only by the creativity of its creators. These solutions are able to embed computational intelligence into different devices, allowing for more practical and useful environments, such as smart homes and offices which have the ability to collect and deliver data to their users.

Unlike other fields that require specific hardware for its development, the barrier to entry for IoT is relatively low. in part due to the prevalence of widely used and mostly open-source prototyping platforms like Arduino [25] and Raspberry Pi [26]. These platforms have been used in various educational initiatives. They assist in solution development, particularly in creating prototypes, through accessible documentation, community support, and a wide selection of modules with integrated libraries.

Arduino has open-source hardware and software, while Raspberry Pi works with open operating systems. These inexpensive and open platforms have either a microcontroller or a microprocessor, input, and output ports for a variety of modules and sensors, as well as the needed power delivery and co-processor on a single board. These single-board solutions aid in prototyping before mass production, where the whole solution is usually reduced to its indispensable components in a custom board. The high popularity of these platforms is also due to the vast component selection, ease of use, and community support as stated by Kondaveeti et al. [27].

These prototyping platforms are also extensively used in education, as shown by Perenc, Jaworski & Duch [28] not only as the first step in robotics or IoT but also as a first step into general programming. Furthermore, many authors, such Schwab & Davis [6] use their influence to incentive non-developers to take part in the 'humanization of things',

which is another initiative that contributes to the technology's high accessibility.

An important concept in this work is a subset of the Internet of Things, the nodes/ client-sided devices, more specifically the cyber-physical devices. Lee [29] describes cyber-physical systems (CPS) by their computational and physical capabilities that can interact with humans and/or other machines by many new modalities. Our work targets these end nodes, physical devices, and things.

2.1.1 Development Platforms

As discussed earlier, there is a lack of standardization for IoT devices, the broadest definition being any device with either input and/or output capabilities, being connected to a network. However, there are prototyping platforms that dominate the market in terms of popularity, such as Arduino[25] and Raspberry[26]. These prototyping platforms are relevant to our discussion once they are representative of real-world applications.

Typically, these platforms comprise a single-board solution with embedded circuitry, predominantly running open-source software and hardware. They facilitate rapid prototyping by connecting to modules, often including sensors within the single-board solution. These platforms reduce the difficulty for new developers to get started with IoT development, with plug-and-play modules and easy programming. These prototyping and learning platforms are not designed to scale and perform at the level of quality required for commercial solutions, where they often end up.

There are many commercial ventures that use these platforms, not only as a prototyping tool, some or all parts such as board design, bootloader, or code executable make their way into their final product. The Creality company has 3D printers named Ender which not only uses off-the-shelf parts but also uses Open-Source firmware.

Raspberry Pi is Severance [30]'s $25 computer, with ARM processors, similar to the ones present in smartphones, it is usually much more powerful than an Arduino board, being capable of running home and server Operating Systems such as many Linux distributions and Microsoft's Windows 10 IoT Core. It is not specialized in IoT. However, it is very popular due to its low cost and compatibility with the Arduino ecosystem.

The closest thing to standardization in the industry is primarily driven by Arduino's dominant position. Where component manufacturers choose to support the platform in order to guarantee sales. Raspberry and other platforms offer native support for any Arduino component or library. Arduino's popularity is often stated in the literature, Kondaveeti et al. [27] in countless areas, from system designs to healthcare, education, mining, and hardware communication. These boards are favored by the maker community being used for homemade projects and prototyping of commercial products. All this success is attributed to the ease of use and low cost Javed [25].

Arduino boards usually contain 8-bit micro-controllers (MCUs) from Atmel, which uses a Reduced Instruction Set Computer (RISC) architecture. Only a few models have 32-bit ARM or x86 processors, Intel has produced the Arduino-compatible Edison, an SoC with two Atom cores. These prototyping boards can be programmed in any language which can be compiled into binary. However, C and C++ are the most popular and the ones supported by the official IDE.

2.1.2 IoT's Software Quality

Software quality is a related aspect of this work, as it necessitates the application of a systematic approach to evaluate the achieved outcomes. The application of software metrics is a way to quantify quality by the selected criteria, as stated by Nuñez-Varela et al. [31]. According to Xenos et al. [32], the choice of software metrics should be closely aligned with the specific project, as the use of inappropriate metrics can render the resulting analysis meaningless.

According to Martin [33], using Object-Oriented Programming alone is not enough to create better software solutions with strong structures, ease of maintainability, and reusability. To achieve these goals, it is necessary to analyze code using software metrics specifically selected for the relationship between objects.

In research about specific methods of IoT presented by Imani et al. [34], determining 10 problematic areas: Interoperability of Standards; Mobility Support; Addressing of Smart Objects; Transport Protocols; QoS Support, Authentication; Data Integrity; Privacy; Security and Digital Forgetting. Ray et al. [35] also note other factors which inflect code quality, such as codebase size, team size, and processes' maturity. Studies which address software quality evaluation when applied to CPD or IoT paradigms are still scarce, and only recently published.

Bures [36] published a paper on the topic of Integration Testing of IoT solutions, proposing a testing framework, and raising the topic of how IoT has specific demands for testing and quality assurance in a number of different areas. The author concludes that these testing methodologies are continuously growing in importance. Bures et al. [37] later published a comprehensive view on quality characteristics of the IoT solutions, more focused on System Under Test scenarios. The authors cite the ISO25010 [1] as a valid set of quality characteristics, they also emphasize the vast differences between IoT and other paradigms. The authors reference various endeavors to classify metrics across diverse areas of IoT. Their objective in this work is to offer a more consolidated perspective, along with a revised list of quality characteristics and accompanying descriptions. Klima et al. [38] mention how the problem of code quality has been addressed in many software studies, proposing an established and consolidated set of metrics for multiple paradigms. It's worth noting that such an extensive effort has not yet been undertaken for IoT. The authors

provide an important discussion on how each code metric relates to software quality in regards to IoT, mapping which code metric impacts, which quality characteristic given by the ISO25010 [1].

To evaluate the studies, a more systematic approach to software quality was necessary. Out of the available quality models, the ISO25010 [1] model was selected, due to its international recognition and high availability of documentation. The ISO 25010:2011 is a revised model of software quality requirements and evaluations, built on an international consensus. The model defines a set of quality attributes, these are categorized into eight characteristics shown in Figure 1. All the problematic areas listed by Imani et al. [34] are covered by the ISO, which reinforces the model's relevance for IoT development.

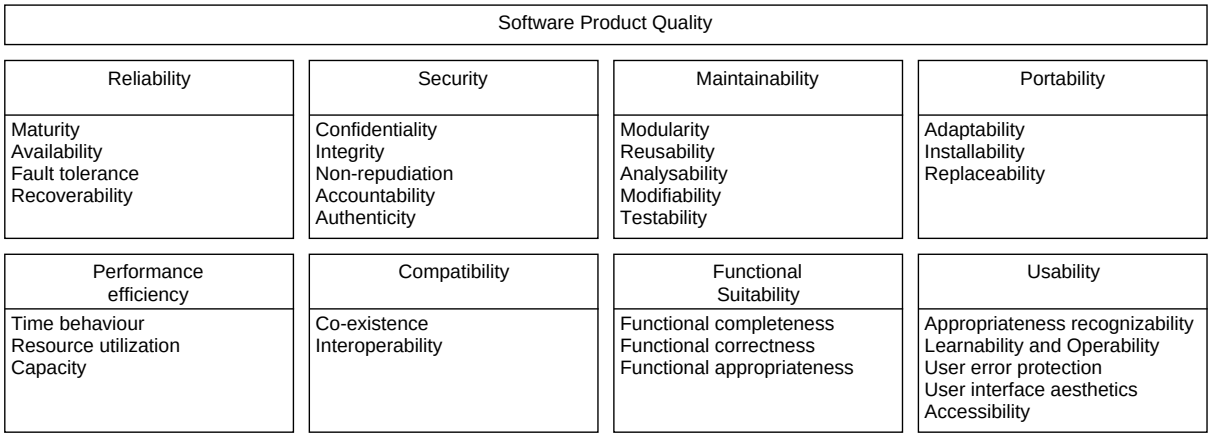


Figure 1 – ISO's eight characteristics ISO25010 [1].

2.2 Model-Driven Development

Model-Driven Development (MDD) is a software development methodology that facilitates the generation of source code from abstracted application models, which results in a smaller gap between the solution and its practical implementation. Stahl, Voelter & Czarnecki [39] compiles a list of objectives for MDD:

- Higher development speed due to automation as executable source code can be extracted from models through transformations.
- Transformations provide better software quality since a selected architecture would be applied to the entire solution.
- Cross-cutting implementations can only be edited at a specific moment, such as code error correction.
- Greater re-usability as architectures can be implemented in a production line for multiple systems.

- Manageable complexity thanks to higher abstraction when editing models, replacing part of code writing.
- Offers an environment that encourages good practices in the following fields: technology, engineering, and management.

Mellor, Clark & Futagami [40] specify a model using MDD as a composite of formal and coherent elements that describe the object, which is favorable for analysis. The higher the level of abstraction, the closer it is to the solution and the further it is from technical language, which enhances comprehension.

A model, as described by Stahl, Voelter & Czarnecki [39] is an abstract representation of a system's structure. These models are usually defined using UML or UML-like diagrams. These diagrams have to be interpreted by a modeling language that applies the transformations' rules.

A meta-model as described by Stahl, Voelter & Czarnecki [39] is one of the main aspects of MDD, aiding in the following challenges:

- In the construction of Domain-Specific Languages (DSLs), describing their abstract syntaxes.
- In the validation of models: the restrictions are specified in the meta-model.
- Model to model transformations, also specified in the meta-model.
- Code generation: by providing reference material for the generation tool.
- Tool integration: tools can be domain specialized by exchanging meta-models.

Meta-models are models with declarations about the modeling, where model structures, their relations, restrictions, and rules are described. It defines the abstract syntax and the static semantics of the modeling language. Higher abstraction models, will inherit from these meta-models.

MDD uses several models at different levels of abstraction, and these models go through transformations until a concrete implementation is reached.

Platform-Independent Model (PIM), as described by Ayed, Delanote & Berbers [10], acks technical specifics, while Platform-Specific Model (PSM) is a representation of the same system but with all technical details necessary for the concrete implementation. The mapping from PIM to PSM is performed by transformations, produced with the aid of an MDD-specific tool, and ultimately, these models are converted into source code.

The definition and separation of PIM and PSM is a key concept for Model-Driven Architecture (MDA), MDA is a subset of MDD, a list of guidelines defined by the Object

Management Group (OMG). It is noted by Stahl, Voelter & Czarnecki [39] that the definition of these models is a relative concept, varying for each platform, lacking a precisely defined abstraction level.

Favre [41] provides the OMG's official and most recent guidelines on MDA. In the document the following three models are defined as such:

1. Computation Independent Model: CIM is a legacy model used to represent business or domain knowledge, modeling real people, places, things, and laws.
2. Platform Independent Model: PIM defines a high-level systems architecture to meet business requirements.
3. Platform-Specific Model: PSM is the result of applying transformation rules to a PIM. This transformation brings the model closer to the specific technology used for implementation.

The official document reinforces the high relativity of these model layers, stating that a PSM is any model that is more technology-specific than the related PIM.

Stahl, Voelter & Czarnecki [39] present reasons for executable code generation with MDD:

- Performance: the authors mention performance as the main incentive for code generation whenever high performance and flexibility are both needed.
- Code size: the length of source code can be easily optimized with code generation, reducing unneeded code, and aiding readability and maintainability.
- Analyzability: from generic to complex frameworks, they all tend to reallocate programming language-related complexity for their own proprietary configuration.
- Early error detection: flexible systems let developers use weakly typed languages, which can cause execution errors that could have been avoided.
- Platform Compatibility: as the programmed logic is separated from the implementation level, it can be easily reused when migrating to newer platforms.
- Language restriction: when generating source code it is possible to circumvent programming language expressiveness restrictions.
- Aspects: cross-cutting properties such as persistence can be implemented in a unified place.
- Introspection: the software's ability to access itself can have its structure generated before execution time.

There are several specific reasons that suggest MDD is particularly well suited for addressing software quality challenges in IoT development, such as:

- MDD presents itself as a great way to reduce complexity, especially in IoT solutions, for its inherently modular development, which is very susceptible to code reuse and recycling [2].
- MDD allows for the generation of abstract models that can be used to generate code for many platforms and technologies, which is important in IoT development due to the wide range of devices and platforms involved [21].
- MDD can improve the quality and reliability of IoT systems, by enabling early verification and validation through model-based testing and simulation [42].
- MDD can promote collaboration and communication among IoT development teams, by providing a common language and set of modeling tools, despite being used to target different platforms.
- Aids the integration of heterogeneous devices: In IoT systems, a variety of devices with different hardware, software, and communication protocols need to work together. MDD can aid to model and generate the necessary interfaces and adapters to integrate these devices seamlessly [20].
- Supports agile development: IoT systems require rapid prototyping and continuous integration to respond to changing user needs. MDD can support agile development by enabling the rapid creation and modification of models that represent the system's structure and behavior [43].
- Enhances system maintainability: IoT systems are often deployed in dynamic and distributed environments, which can lead to system failures and maintenance challenges. MDD can facilitate the maintenance of IoT systems by providing high-level abstractions that can be used to reason about the system's behavior and diagnose faults [44].

In this section, related work is discussed with the intent of contrasting it with the proposed framework. Included works must contain platforms, frameworks, methodology, and tools that focus on or include on-device Cyber-Physical devices development.

2.3 Systematic Mapping

In this section we briefly describe the main findings from our Systematic Mapping [45] which was the initial artifact used for gathering related work, the full paper is attached

as appendix C, it includes more in-depth details about the methodology, full list of research questions, search string, study selection, and all other relevant information.

The purpose of this mapping study is to investigate the extent to which software quality is prioritized by IoT developers, what methodologies are applied, and how software quality influences IoT development in client-side implementations. Following the guidelines from Kitchenham et al. [46] and Petersen et al. [47].

The Snowballing technique was applied in the first revision of the Systematic Mapping. Subsequent revisions required modifications to the search string, which were broadened to encompass all papers originally identified through the Snowballing process.

RQ01: How often is every quality requirement cited in the analyzed papers?

The studies were categorized based on the quality characteristics they discussed. These characteristics were either mentioned only briefly as a side effect of the study or were a direct focus of the work being proposed. In Figure 2, the quality characteristics by year are presented and it shows a significant increase for specific characteristics such as Security and Performance. The quality characteristic classification was executed during the full paper reading phase.

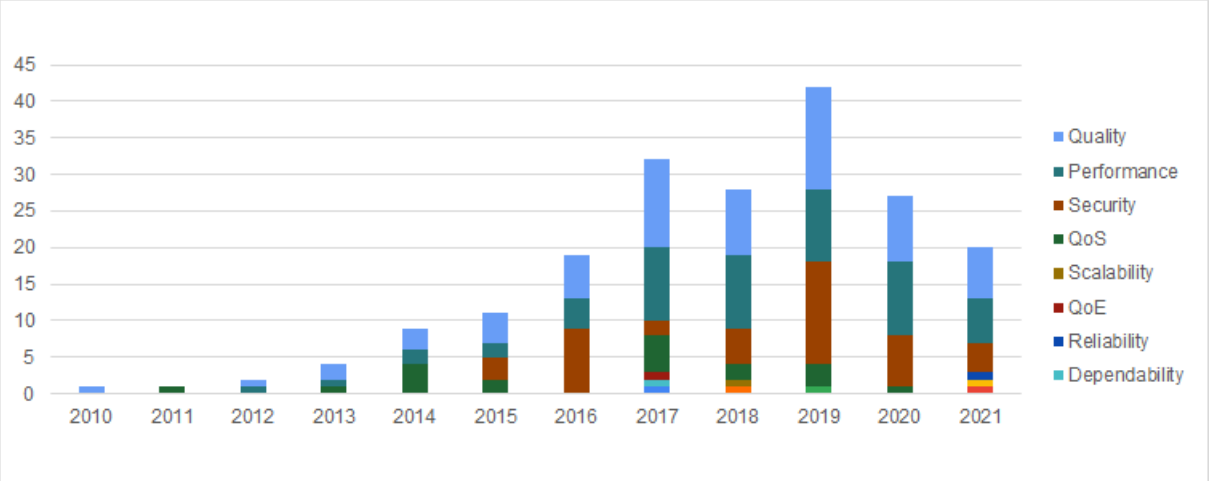


Figure 2 – Quality characteristic by year.

Whenever possible the studies were classified by the quality characteristics as defined by the ISO25010 ISO25010 [1]. However, many studies did not use one of the ISO's standardized terms, leaving to the authors of this work to categorize them, notable mentions include Quality of Service (QoS), which can bear different meanings, such when regarding the overall performance or to traffic prioritization in the networking context. "Quality" refers to studies when they did not specify which quality characteristic is being considered.

It was possible to plot a bubble chart of quality characteristics against the study domain(Figure 3) (relations with less than 2 items per row/column were removed in order to maintain readability). We determined the study domain by the study's primary objective, for instance, if the study proposed an implementation focused on improving communication from a client-side device, its domain is defined as "Communication". As expected the biggest correlations were where the study objective matched the mentioned quality characteristic. Less obvious information obtained from the chart was the distributed mention of performance.

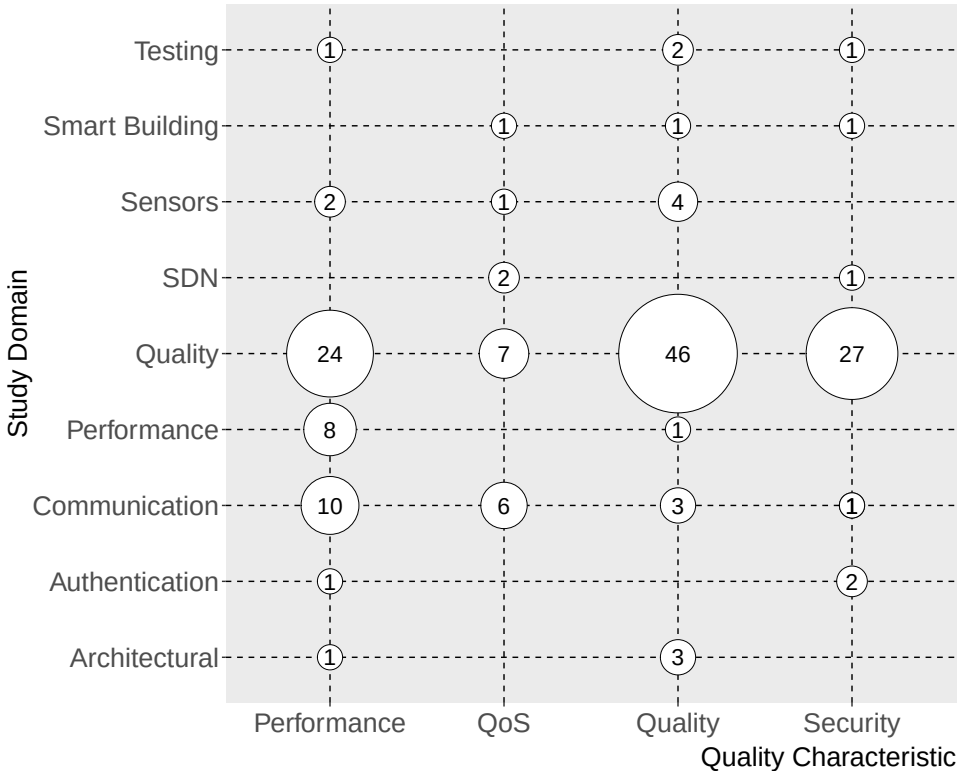


Figure 3 – Study domain as contrasted by quality characteristics.

Answering this research question directly, 96 out of the 211 studies cited being conscious about a specific quality characteristic. Out of the 96, 45.92% of the studies had quality-related domains. The most popular characteristics were quality, security, performance, and QoS, all other quality characteristics were seldom found.

RQ04: Which platform, domain, framework, and/or paradigm is being used?

This extraction had the most open-ended field, requiring a greater effort in filtering and grouping information. As result satisfying an exploratory objective of what could be mapped. We gathered hardware and software platforms and frameworks, different communication techniques, database domains, module specific solutions. Figure 4 represents them by year, notable trends are the growing focus on quality and security studies.

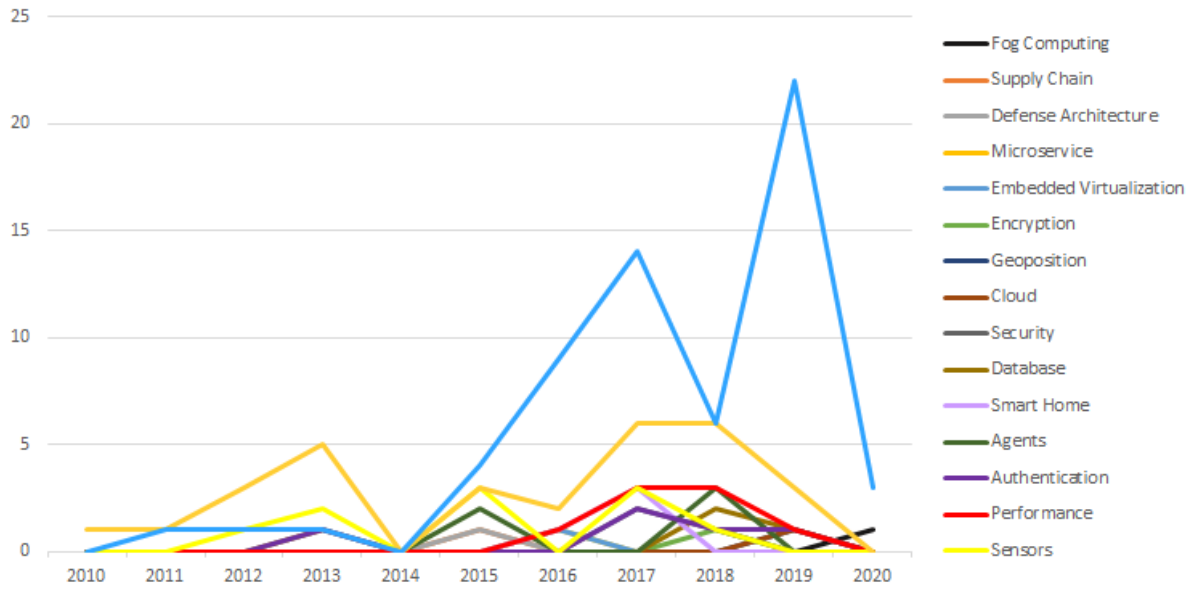


Figure 4 – Studies' domain.

Systematic Mapping Conclusion

During the systematic mapping process, we conducted searches in major publications to identify relevant studies on software quality in the field of IoT. Our findings revealed an increase in the number of publications related to IoT, but we also identified significant gaps in the area of client-side development for IoT.

The main contribution of the study, in relation to the current chapter, was aided by the initial assessment of techniques and tools used on cyber-physical devices, some of which are detailed in the next section.

2.4 Related Work

Attributable to the list of advantages Model-Based solutions offer and, especially when applied to a highly modularized paradigm such as IoT, there is plenty of work that implements the methodology, usually as Model-Driven Development. However, solutions that implement different methods and techniques are also analyzed. Concurrent solutions are recently plentiful and growing in quantity.

The remaining works explored in this section do not use Model-Based Approaches whilst having the same objectives as the proposed solution. They are mostly Low-Code or No-Code approaches, which have usually programmed steps replaced with automatically generated or graphically edited steps (these definitions may include Model-Based).

Nguyen et al. [2] proposed the Framework for Sensor Application Development (FRASAD). FRASAD intends to create a framework capable of coping with the hetero-

geneity and complexity of IoT sensors and systems. Figure 6 illustrates how it does not have any elements representing decision-making, the authors explain that the logical conditions are configured with specific and proprietary flags on an additional development step.

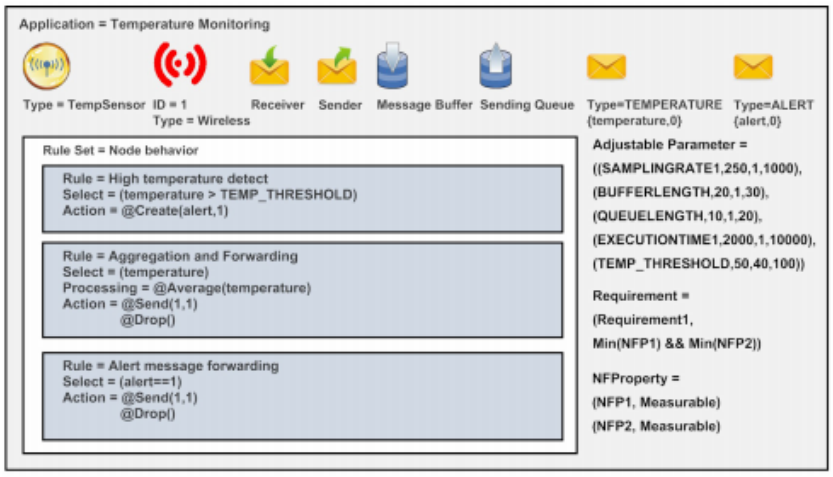


Figure 5 – PIM Sample provided by the author (Source: [2, p. 4]).

The user interface in Figure 5 shows that decision-making is mostly done in a text-based manner, which can be quite verbose. Additionally, it includes boilerplate code that requires the user to enter the correct methods in the model. Our framework plans to unify these development steps in a single visual step.

Currently, the framework supports only models based on a DSL meta-model. The authors state the availability of their software as an Eclipse IDE plugin, as of the publication date, it appears to be inaccessible.

Ciccozzi et al. [48] defines MDE (Model-Driven Engineering) as a key enabling agent when the topic is IoT. The authors propose an approach named MDE4IoT and focus on the auto-adaptation of systems and devices present in IoT. The method was tested using the "Smart Lights" case study, where all the subsystems are connected by a network. If any of the traffic lights fail, the MDE4IoT handles the situation in real time, as outlined in the meta-model. Although this method could be applied with tools, currently it is only a research methodology.

The UML4IoT was introduced by FLOCH [49] and its purpose is to connect older mechanical components with the current IoT-based Industry 4.0 by using a UML profile. Its profile is used in a model-driven approach for the automatic generation of an intermediate layer called IoTwrapper, requiring additional hardware.

NETIoT was developed by Pantelimon et al. [50] and is one of the many Low-Code approaches, and its main merit is offering a development platform with zero intervention

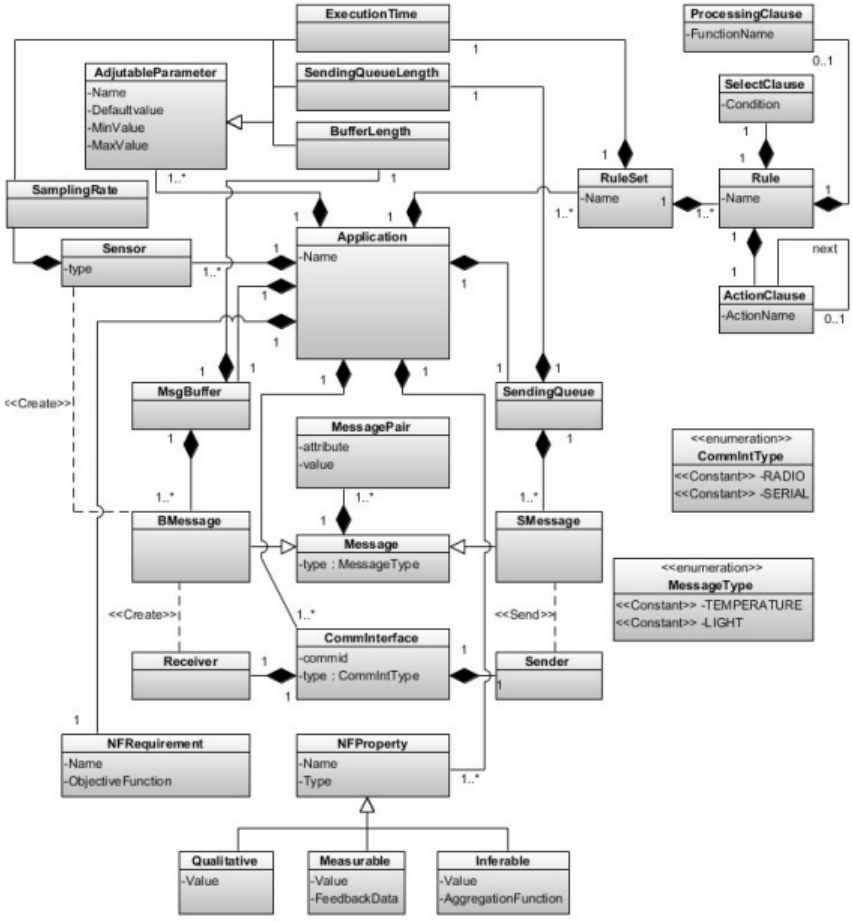


Figure 6 – FRASAD's diagram for a DSL meta-model (Source: [2, p. 3]).

on hardware and minimum configuration effort. The platform has a large scope, generating a solution for the whole paradigm and not only the hardware-node devices. By using JavaScript the author was able to offer an easier learning curve for developers at a cost of marginally lower execution performance when compared to C++. The given example is limited to encoding and decoding protocol.

Autodesk's TinkerCAD has a 'Circuits' page with Arduinos and components, where you can connect schematics and simulate execution. It is an example of many similar platforms targeted to beginner programmers and students, code input can be made with coding 'blocks', a concept popularized by Scratch. The code generation, however, does not make use of C++ capabilities such as Object Oriented Programming.

Created by IBM, Node-RED is an open-source flow-based programming tool used for the visual programming of IoT devices. It is a web-based platform that provides an interface for connecting different devices, APIs, and online services to build custom applications. Node-RED uses a drag-and-drop interface, shown in Figure 8, that allows users to create flows, where each node represents a specific task or operation. It is an example of a platform that targets IoT developers and enthusiasts, and that provides a

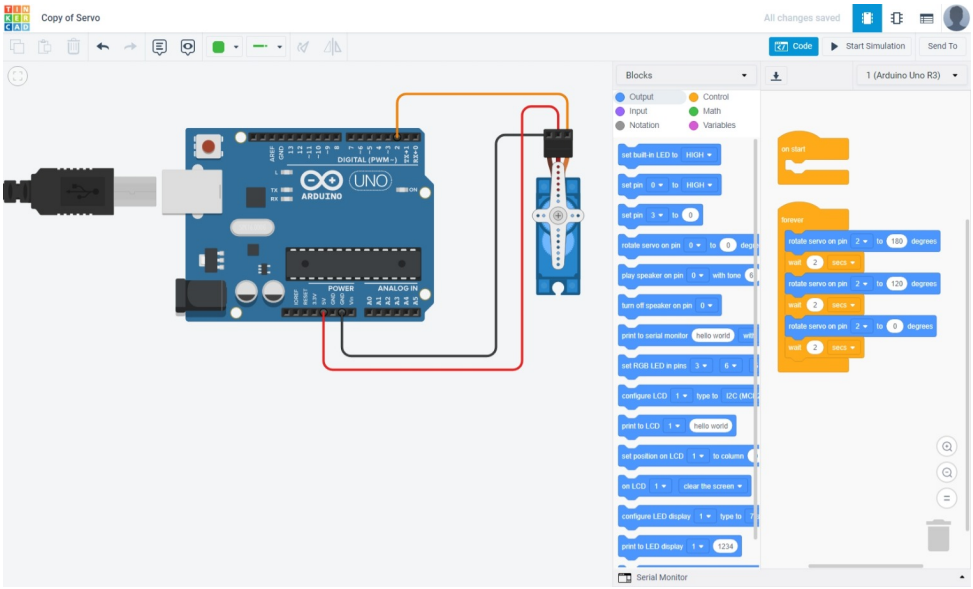


Figure 7 – Tinkercad's blocks interface.

set of pre-built nodes that can be customized to perform various tasks.

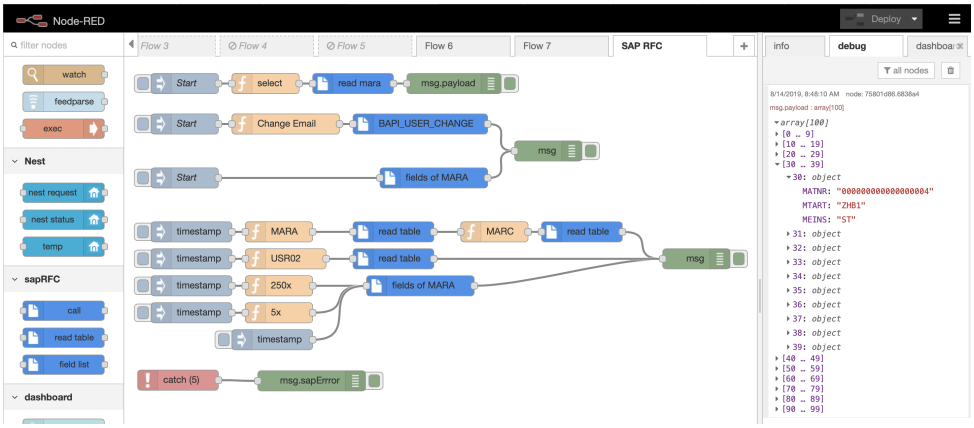


Figure 8 – Node-RED's web interface.

Although it does not use conventional programming languages, Node-RED provides a versatile set of nodes that can be used to create complex IoT applications without the need for extensive programming knowledge for prototyping-level work.

Some of the platform's limitations are discussed by Thuluva et al. [51], where is concluded that it is not suitable for production, due to it not being scalable nor performatic.

2.5 Considerations

Despite the existence of multiple works proposing different solutions for aiding developers' work on Cyber-Physical devices, none has been found with all qualities of

the framework here proposed. The most notable differences are the lack of execution performance loss, the extensibility of the model, and the edit-ability of the generated code.

3 RESEARCH STRUCTURE

This chapter describes the structure of the research followed by the methodology. This work is divided into 9 steps distributed over 4 stages, as illustrated in Figure 9, where a solid line indicates strict flow and a dashed line indicates an optional/referential flow, each stage is defined and explained. This study is characterized as applied research with the goal of both creating new knowledge and producing practical applications.

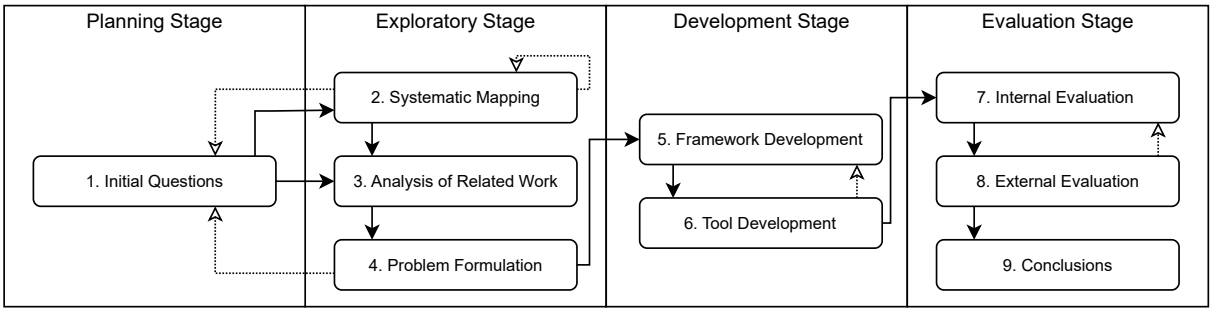


Figure 9 – Research structure.

3.1 Planning Stage

The Planning Stage is composed exclusively of the Initial Questions step, responsible for the initial data and knowledge acquisition required for setting up the scope and pathway of the research. The research work was guided by establishing initial questions, such as: “What frameworks, standards, platforms, and guidelines are developers using for software development?”, “What tools are being used for maintaining software quality?”, and “What criteria are being used to judge software quality?”. These questions were asked in the IoT/cyber-physical development field.

These questions are used as an initial guide and being susceptible to changes. They were needed for the research scope setting.

3.2 Exploratory Stage

The exploratory stage is where most of the external research and information gathered is performed, the discoveries made in this stage were capable of causing the revisiting of the previous stage, adequating the research questions accordingly. This stage was composed of the following steps: Systematic Mapping, Analysis of Related Work, and Problem Formulation.

A systematic mapping, folllowing the guidelines by Petersen, Vakkalanka & Kuzniarz [52], was conducted as the entry point to the Exploratory Stage. The mapping results were partially unexpected. For example, despite the number of relevant studies being large and increasing every year, none of the studies stated the software metrics or tools used in analyzing software quality, nor the programming paradigm applied. The Snowballing technique was applied on the first revision of the Systematic Mapping. Subsequent revisions required modifications to the search string, which were broadened to encompass all papers originally identified through the Snowballing process.

The ongoing process of analyzing related works has resulted in a sizable collection of studies, many of them were found during the Systematic Mapping step. Due to the suitability of MDD for highly modularized paradigms like IoT, there is a wealth of literature that employs this methodology within the IoT domain.

An overlapping of MDD and low-code solutions was observed. All of the related work contains limitations which were discussed in Section 2.4. There has been a significant increase in the quantity of no-code or low-code solutions addressing IoT in recent years, accompanying the growth of these solutions in general.

Especially on MDD solutions, none of them has feature-parity with our proposed framework, lacking a hardware-based model abstraction, the ability of editing generated models, and not having performance overhead.

3.3 Development Stage

The proposed framework contains some models which use similar abstraction layers to OMG's MDA. However, it has unprecedented elements, such as the abstraction derived directly from hardware components, containing its metadata, and source code extracted from components' libraries.

The framework was developed using Model-Driven Development as the basis, with the objective of managing different inputs into executable and editable source code. The inputs to the system are the boiler plate code, the meta-model provided by the tool maintainer, the device library provided by the component manufacturer, and the graphical modelling done by the IoT developer in a UML-inspired diagram defined by the authors.

The tool was developed in parallel with the framework. The tool provides a visual editing environment, and an interpreter triggered by diagram changes, which transforms the diagram into working source-code. The tool development took place using Agile's definition for Design Sprints, using particularly short sprints, due to the requirement for rapid review and refinement to the framework specifications.

3.4 Evaluation Stage

The evaluation was conducted in two distinct steps, the first one is described as Internal Evaluation (step 7) and was conducted by following the guidelines from Wohlin et al. [53], demanding an empirical strategy for performing a quasi-experiment. The second step is an External Evaluation discussed in the next section.

The Internal Evaluation determines the viability of the proposed framework in aiding software development, by analyzing the quality of the produced solutions. Execution time analysis and software metrics were used in order to compare the framework-produced solutions to traditionally made ones. We were able to present the analysis of the software generated by the proposed framework, the viability status of its implementation, and the resulting software quality.

The External Evaluation (step 8), was set out to evaluate the perception of usability of the framework's tool, for this, the Technology Acceptance Model was applied to 14 test subjects. The test subjects were working professionals with distinct experience levels, they were observed while reading a user manual and trying to reproduce a given solution using a prototype version o the framework's tool. None of the subjects had any prior contact with the framework or tool.

4 MDD4CPD DEVELOPMENT

This chapter is divided in two sections, where the proposed MDD4CPD framework and tool are explored, MDD4CPD has the objective to improve software development processes' quality and speed, lower domain knowledge needs, and error production, by using a tool that takes some responsibility away from the developer.

The framework brings a novel approach to MDD. Where the abstraction layer structured around the hardware components. A tool was created for supporting the framework's purpose, aiding developers to apply MDD in IoT development, providing the developer a user interface capable of visual editing an abstract software model. The tool would then generate executable code that is also well structured and user readable.

4.1 MDD4CPD Framework

The proposed framework targets client-sided implementations, which will be referred as cyber-physical devices. Further IoT nomenclature is discussed in the Background chapter. Figure 10 shows the framework's workflow:

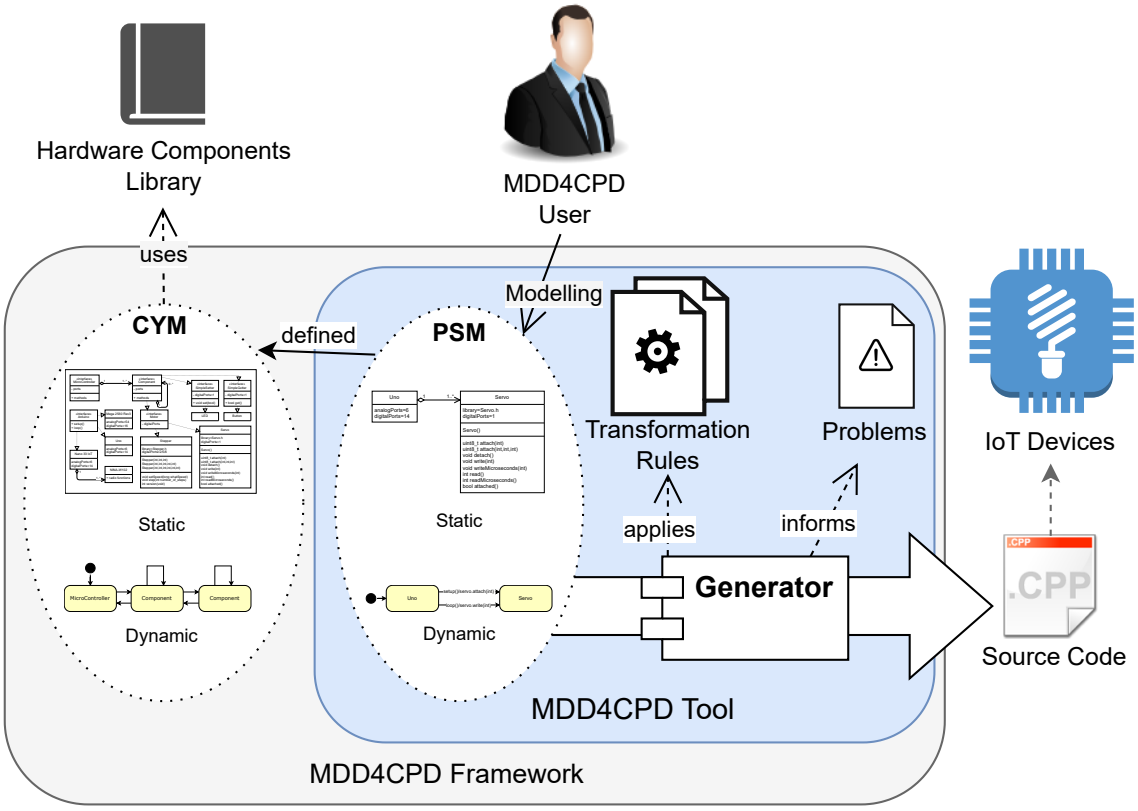


Figure 10 – MDD4CPD's Framework.

1. Before the framework user can start modeling, some artifacts created by the framework maintainer must already be in place, these are the Static and Dynamic CYMs, and the transformation rules.

2. The framework user can then start modeling the PSM by visual editing inside the MDD4CPD tool's GUI.

3. The final step is performed automatically by the tool, it is a complex system that will be explained further in the next section. Briefly, the generator component analyses the PSM at every diagram change; applies the transformations rules to produce source-code, which is ready to be executed on a cyber-physical device. The generator's custom linter will also generate relevant errors and warnings.

The framework lacks the presence of a CIM, the pure business model used in MDA implementations where additional system design decisions are needed to generate the PIM, which was not needed in the MDD4CPD context.

4.1.1 Cyber-Physical Model - CYM

One significant contribution of this work is its innovative method of using a hardware-based metamodel, the Cyber-Physical Model (CYM). It replaces the PIM as an enhancement when contrasted with an MDA solution. This approach differs from traditional MDD solutions that use generic models and diagrams. Therefore, the CYM model was created to address the unique requirements of IoT systems, allowing for more effective modeling and development.

In the cyber-physical domain, CYM provides several benefits compared to the PIM. Briefly speaking, the differences include:

- Each CYM specifies and is mapped to a single hardware component, and it uses the component's documentation for its specification, the documentation includes hardware-specific metadata and library source code. This documentation is inputted before any transformations as it would have been in an MDA implementation;

- By OMG's definition of platform Favre [41], CYM won't be platform-independent once we are defining the working scope on cyber-physical devices;

- The model is presented as a two-part solution, needing a static and a dynamic representation.

However, CYM inherits other aspects from PIM. Figure 11 shows the generic static CYM, which derives from the hardware and documentation elements. There is a one-to-one mapping between a hardware device and the CYM that both represents and controls

it. It uses the Composite Design Pattern since every single class can be represented as being inherited from the same abstract class, as they have shared behavior.

In this abstraction, a 'MicroController' is an aggregation of one or more components, which may have components of their own. 'MicroController' and 'HardwareComponent's have ports, which are hardware addressable general-purpose input/output (GPIO) signal pins.

All components have methods, which will be imported from their respective libraries if provided. The non-hardware based 'Component' is extended to create helper interfaces that will generate all non-hardware-based components, such as software-only libraries for data processing and custom functions.

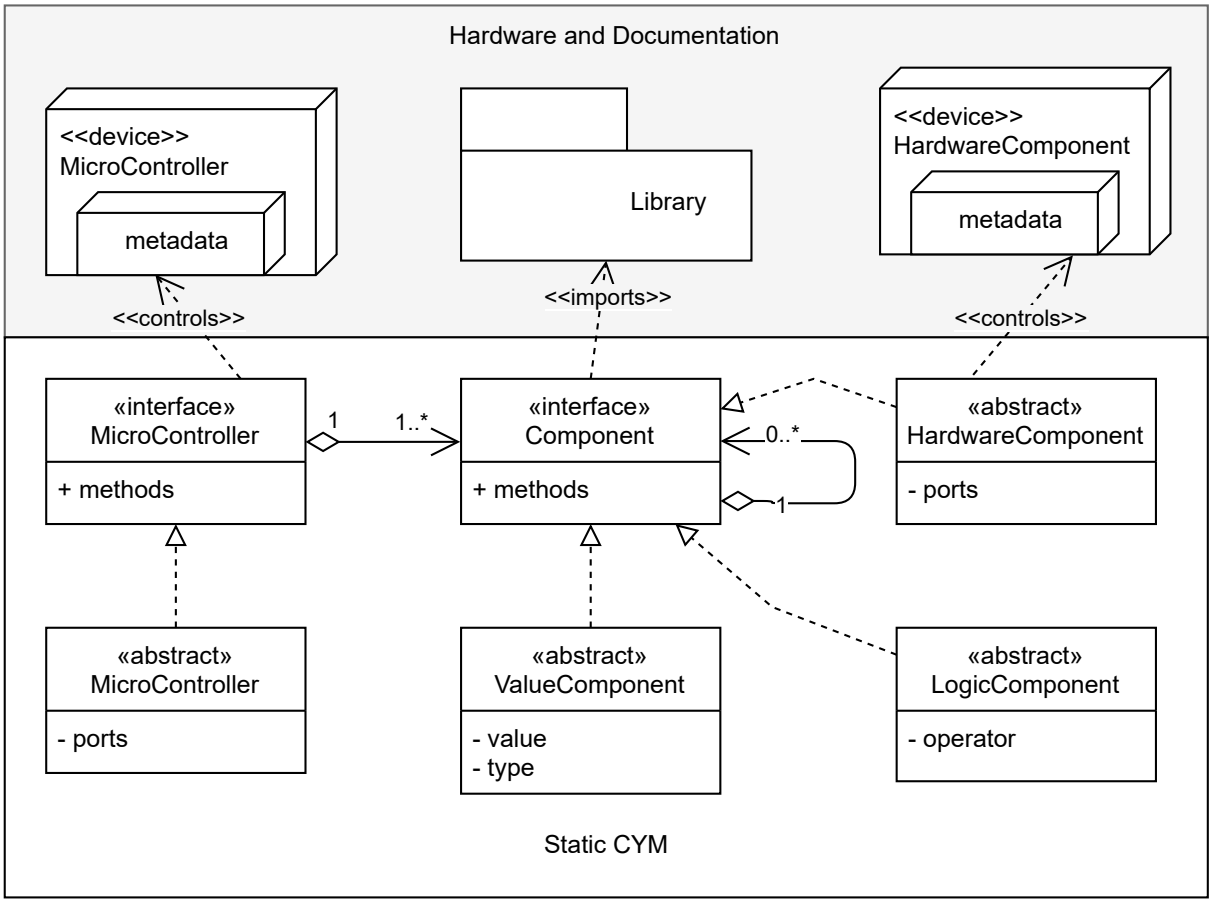


Figure 11 – The hardware-based static CYM.

The presented model was designed to be generic enough to cover the cyber-physical development domain. It was created with the objective of being flexible and extendable by any end-user, in order to bear custom elements or methodologies, as discussed later in this section.

We provided an extended version of the static CYM in Figure 12. It was extended to include a few Arduino micro-controllers; value and logic components; and hardware

components extracted from official libraries. This implementation is sufficient for the comprehensive testing conducted in this work.

A potential class that implements the “MicroController” interface will have specific definitions such as the number of available digital and analog ports as integers, and methods, in the case of an Arduino setup and loop, necessary methods for the Arduino’s lifecycle.

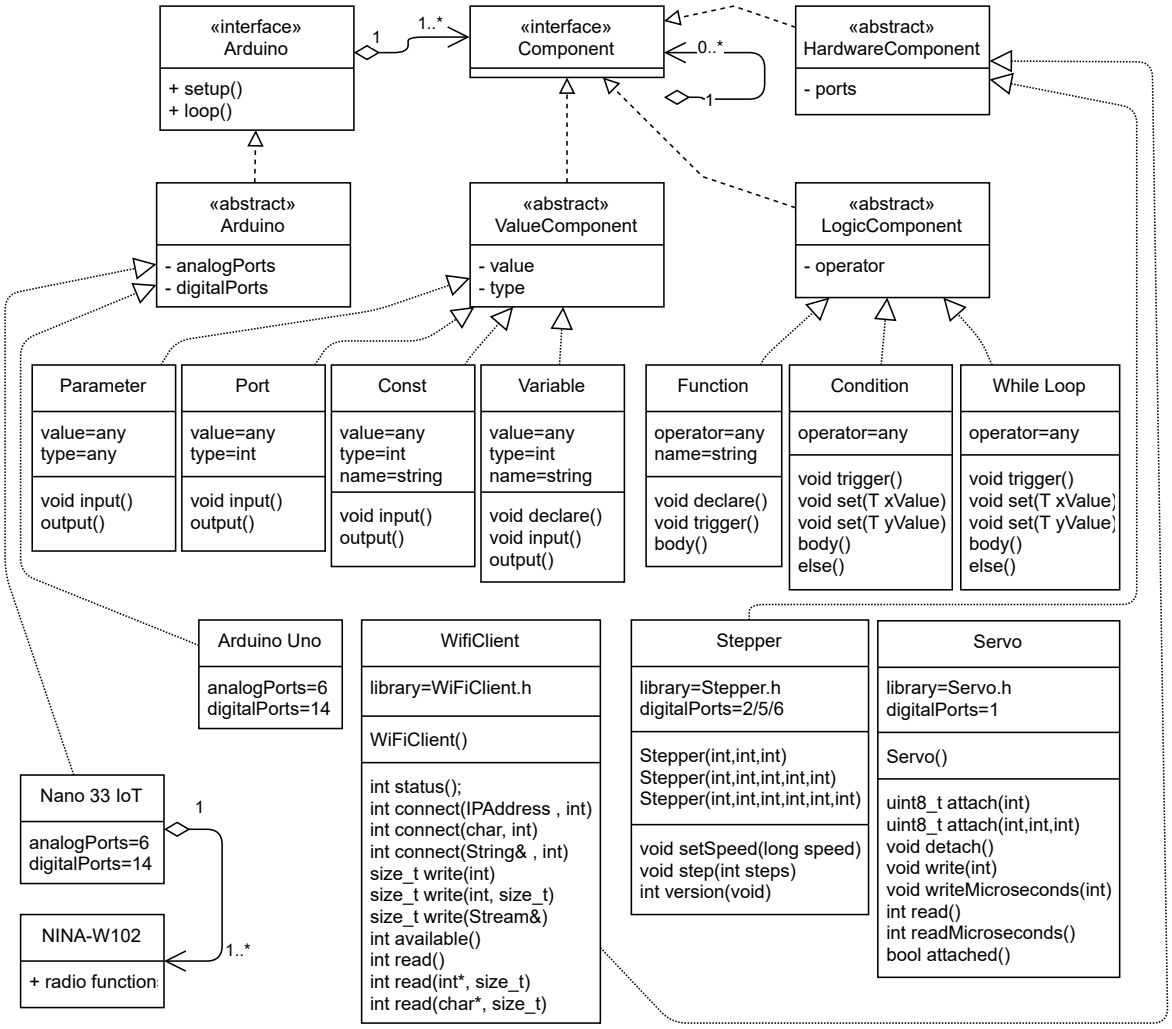


Figure 12 – A static CYM extension for Arduinos.

Additional design decisions were made with the aim of enhancing the alignment of the editing workflow with the restrictions of CYM in mind. For example, only one micro-controller is allowed in the diagram, and every component present is automatically instantiated by the micro-controller, not needing any extra input from the developer.

The static CYM presented can only provide definitions for classes, attributes, and relations, lacking definitions for component decision-making or programming logic, for a complete implementation. The dynamic CYM was designed, this can be seen as a modified

state machine in Figure 13. It holds the definition for the event/trigger for the conditional method calling for each. It has no terminator as the Arduino has no way to turn itself off by software.

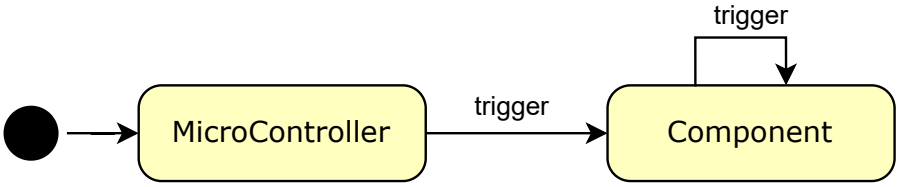


Figure 13 – The hardware-based dynamic CYM.

The capability of one component to trigger another is going to be essential for enabling the chaining of multiple components during the modeling process.

4.1.2 Platform-Specific Model PSM

As defined by the static and dynamic CYMs, the PSM has corresponding static PSM and dynamic PSM abstractions. The static model holds the class definitions for the instantiated components and their relationships, for example in Figure 14 where an Arduino Uno has an aggregation of a Servo (motor type), and that is all it can be extracted by this model.

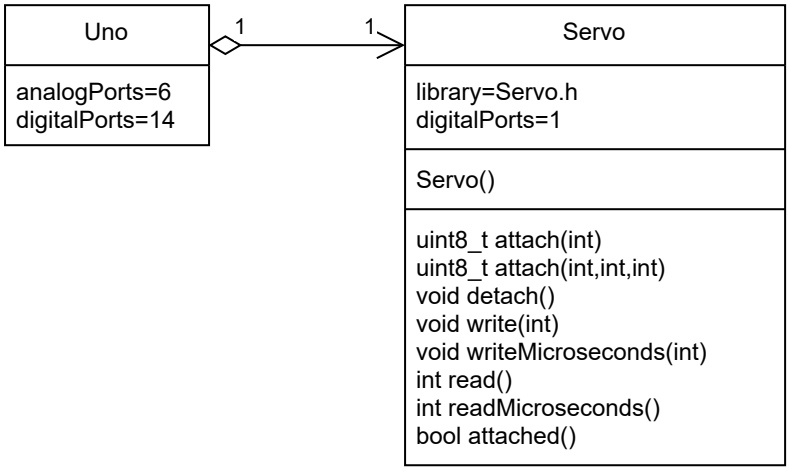


Figure 14 – Static PSM.

The dynamic PSM for the same example has the complementary and necessary information, shown in Figure 15, it describes the life-cycle of the program, where the Uno calls the servo's attach method (pseudo-instantiation, setting the physical port) at setup and then proceeds to call the write (set motor position to a chosen degree) method at each loop call.

Many of the related works presented this segmented abstraction in separated steps into their approaches. They usually have a generalized "classes" step before a "decision-

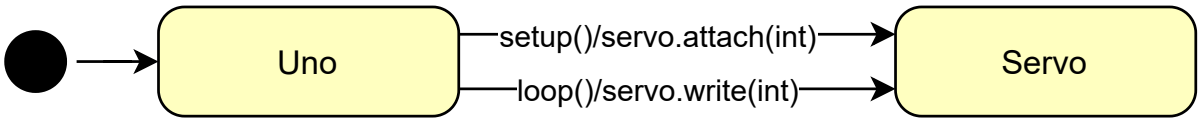


Figure 15 – Dynamic PSM.

taking" step, which usually involves a custom textual language, requiring extensive referencing to the previous step. Another relevant contribution from the MDD4CPD framework is an unified PSM modeling step.

An example of the unified modeling step is shown in Figure 16, it uses the simple "Uno-Servo" example used throughout this subsection, the unified diagram has its both static PSM (Figure 14) and dynamic PSM (Figure 14) replaced by the modeling example provided in Figure 16. The diagram is UML-like, meaning it has UML class diagrams such as attributes and methods, and elements from activity diagrams such as the ones needed for decision-making.

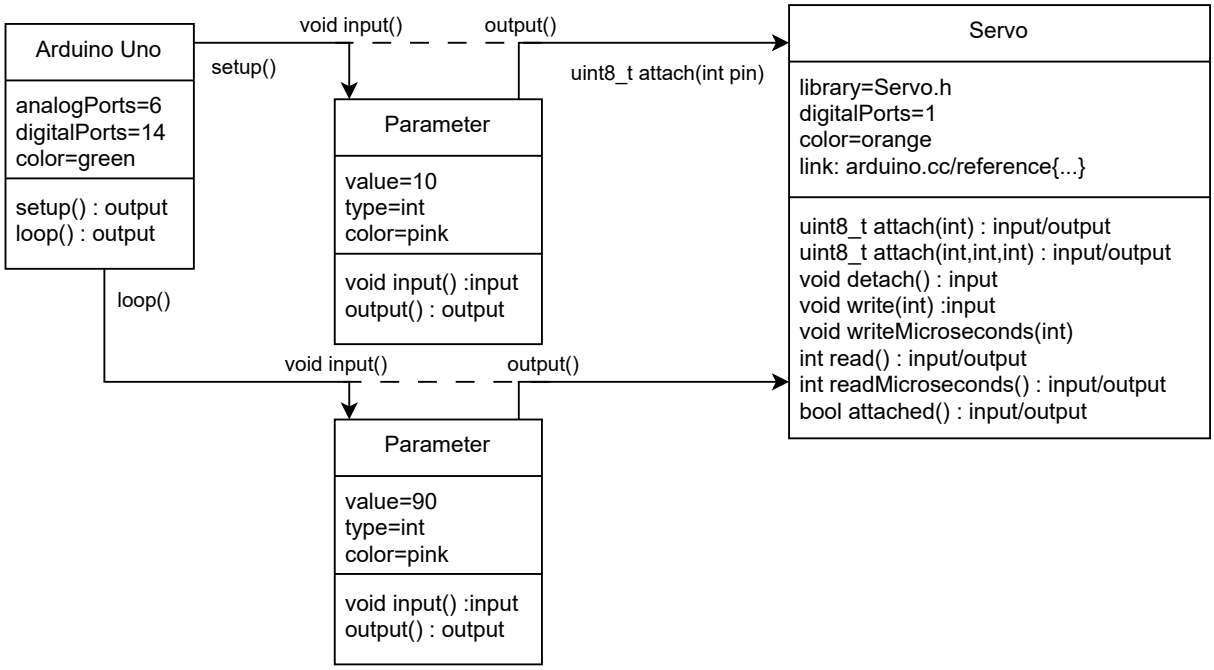


Figure 16 – Complete PSM Modeling using MDD4CPD.

This UML-like representation is done via a custom diagram language The domain-specific language (DSL) is specified in detail in the next subsection. It is noted that diagrams using the DSL, such as the one in Figure 16 are fairly verbose due to having extra elements which are planned to be hidden or alternatively shown by the developed tool's user interface.

4.1.3 Domain-Specific Language

Due to custom attributes presented by the PSM, a custom diagram language was developed. The UML-like characteristic previously discussed is kept, as the diagram will hold both class-level attributes and methods and state-changing information. There are two distinct elements inside the diagram, **nodes** and **links**.

The diagrams displayed in this subsection use the developed DSL, accompanied by the information schema in the JavaScript Object Notation (JSON) format, this is because the entire object that will be fed into the MDD4CPD tool, where the modeling process will occur, is composed of these abstractions.

Nodes

Nodes are deemed required for representing components, in this MDD4CPD's abstraction a micro-controller implements a component interface. Nodes are derived from the static CYM detailed in the previous section. Holding the same values present in the Static CYM, attributes, and methods.

For instance, the following is the real and total information schema used for the Arduino Uno modeling, which will be used to generate the equivalent node in Figure 17:

```
{
    name: 'Arduino Uno', // Display name
    color: 'green', // Background display color
    type: 'controller', // Implemented interface
    extras: { // Micro-controller specific attributes
        analogPorts: 6,
        digitalPorts: 14,
    },
    methods: [ ], // List of methods with Input and Output
    outs: [ // List of methods with only Output
        'setup()',
        'loop()',
    ],
    ins: [ ] // List of methods with only Input
}
```

Methods can be either generic or sorted by "ins/outs", in order for the diagram to assess what methods can be accessed by Input and/or Output. Their meaning is as follows:

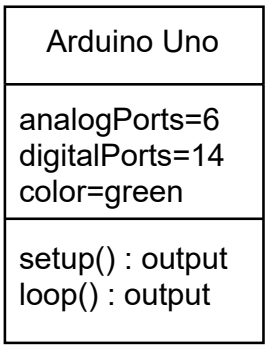


Figure 17 – Arduino Node representation in MDD4CPD's PSM.

- Methods for Input: these are all void returning methods. Such as 'void detach()', this method will be able to triggered only by its Input, not providing a returned value to be used elsewhere.
- Methods for Output: these are methods that can not be triggered by the Input since result does not return a value. Such is the Arduino's lifecycle methods 'setup()' and 'loop()'. Note that these methods are always recognized in the DSL without any kind of return (different to void returning functions).
- Methods for both Input and Output: these will be triggered by their Input, and will provide a returned value in their Output. Such as an 'int read()'.

Analyzing a 'HardwareComponent', the Servo's schema, derived by the officially provided Servo.h library:

```
name: 'Servo',
color: 'orange',
type: 'component',
extras: {
    library: 'Servo.h', // Library entry point
    link: 'arduino.cc/reference/en/libraries/servo/'
    // URL to external documentation
},
methods: [ // Methods provided by the library entry point
    'uint8_t attach(int pin)',
    'uint8_t attach(int pin, int min, int max)',
    'void detach()',
    'void write(int value)',
    'void writeMicroseconds(int value)',
    'int read()',
```

```
        'int readMicroseconds()',
        'bool attached()',
    ]
```

In MDD4CPD's DSL, this Servo component is displayed in Figure 18, it can be noticed how the methods that return void do not provide an Output for its value to be used, only an Input port for the method to be called which may or not include parameters.

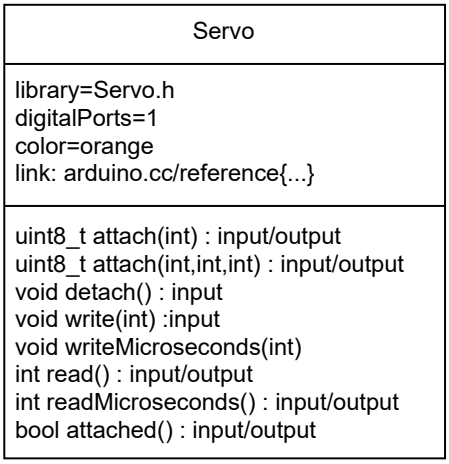


Figure 18 – Servo Node representation in MDD4CPD's PSM.

Besides providing an URL with documentation for users, this node also refers a library entry point file, this file has been parsed manually to get the methods with return types and parameters, library files could be automatically parsed.

Certain information found in the schema and DSL is not intended to be presented as text within the tool, such as color and documentation link, these are to be graphically shown. For instance, the external URL documentation is provided to the user as reference material, and the number of analog and digital ports of a micro-controller is used in order to generate user warnings, during the automatic transformations.

There are some nodes that are not derived from hardware components, these are custom abstraction artifacts used for modeling, Figure 19 shows the three logic nodes which are described below:

- 'Function' is a logic node used for code reuse and organization; it has a function 'name' field; a 'void declare()' Input (when not used, will declare the function in the global scope); an 'void trigger()' Input, which will be used to trigger the function; and a 'body()' Output, which will be used for that function calls. A current limitation of this custom node is it only being able to receive optional parameters, but not provide a return value, these functions will all return void.

- ‘Condition’ is another logic node, used for if or if/else statements, these nodes can be chained; it as an ‘comparison’ selection field containing all C++ comparison operators (==,!=,>,<,>=,<=); a ’void trigger()’ Input, which will be used to trigger the condition; two Inputs for setting the values for the comparison; a ‘body()’ Output port, which will be used in case the comparison statement returns true; an optional ‘else()’ Output port which can be used in case the comparison statement returns false.
- ‘While Loop’ node consists of the exact same configuration as the ‘Condition’ node; but its use will result in repetitions till the first time the comparison statement returns false.

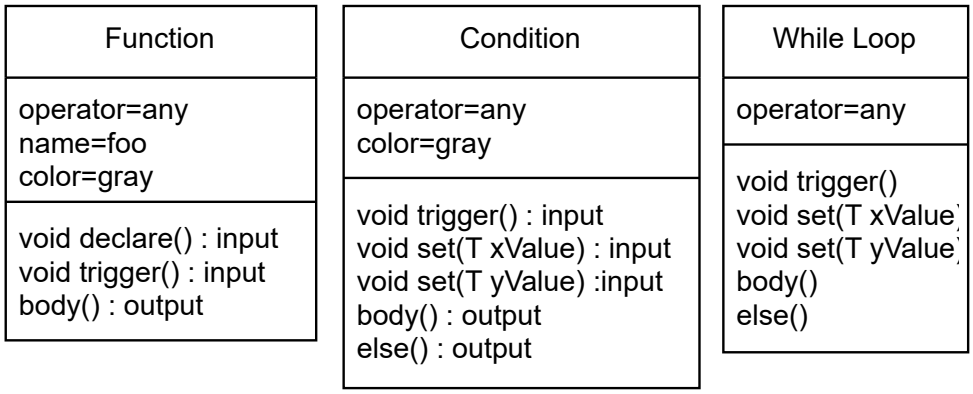


Figure 19 – Custom logic nodes.

Besides the logic nodes, the second provided set of non-hardware based nodes are used for representing values in the model, the four value nodes are shown on Figure 20 and described below:

- ‘Parameter’ node represent unnamed values that can be passed as parameter in custom functions or component methods; these have a ‘type’ selector that is used for editor warnings only; a ‘void input()’ input to be used as a trigger; and a ‘output()’ Output to be linked into the desired function/method, or to be chained into other value nodes if needed.
- ‘Port’ node is just a ‘Parameter’ node with some syntax sugar, it has a (to be hidden) fixed ‘type’ int; an Analog/Digital selector and port number selector that goes from 0 to the number of Analog/Digital ports present in the micro-controller present in the diagram, this is used for conflict warning.
- ‘Constant’ node represents a re-usable value, it has all elements as the ‘Parameter’ node; with an additional ‘name’ input; constants are always declared on the global scope.

- ‘Variable’ node represents another re-usable value, it has all the elements in the ‘Constant’ node; with an additional ‘void declare()’ Input to define the variable scope, if not used it will be declared in the global scope. As a current limitation of this custom node, it is lacking the ability to change its value after the initial declaration.

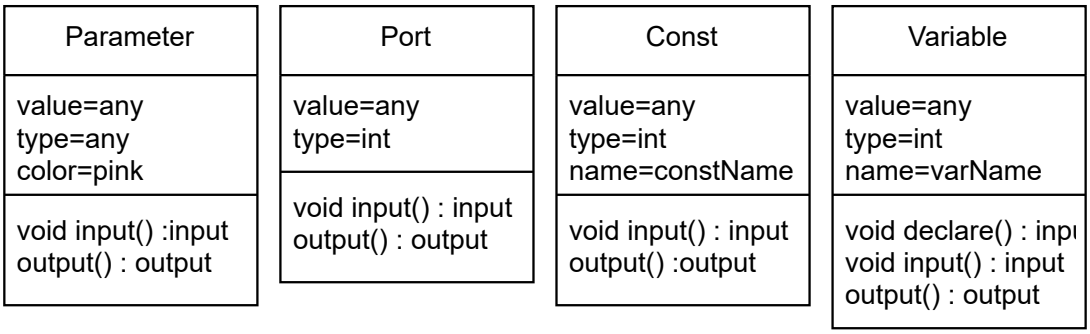


Figure 20 – Custom value nodes.

Links

Links are relations between methods of different nodes, which may carry values, either as parameters or as the returned value. Links are derived from the dynamic CYM, representing the state change in the generated solution. Figures 21 and 22 show links being used to trigger functions without the passing of any data, their only goal is to call either a component method or a custom function.

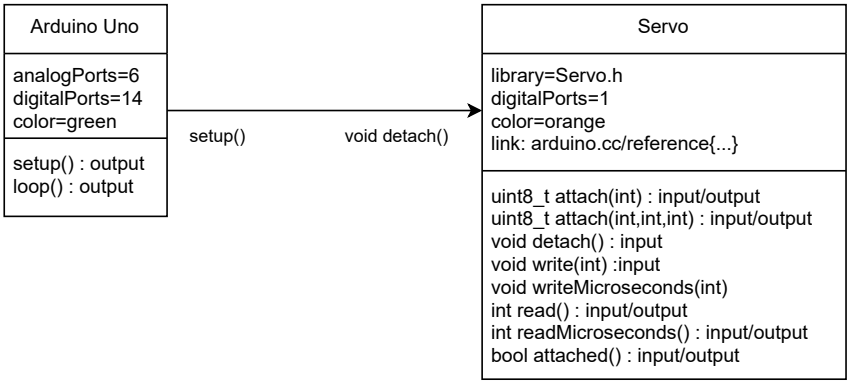


Figure 21 – Link to call the Servo’s detach function.

Alternatively, a link may be used to call a method while providing values as parameters, for instance in Figure 23 the ‘Parameter’ node is part of the link between ‘setup()’ and ‘uint8_t attach()’, its value will be passed onto that method as a parameter.

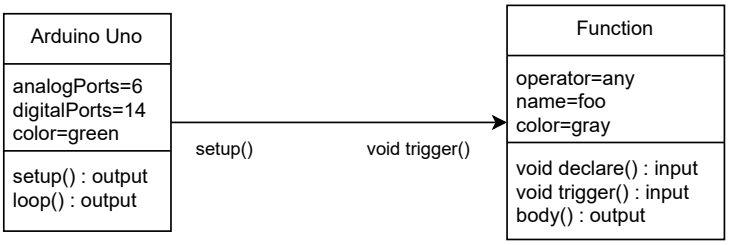


Figure 22 – Link to call the parameterless 'foo' function.

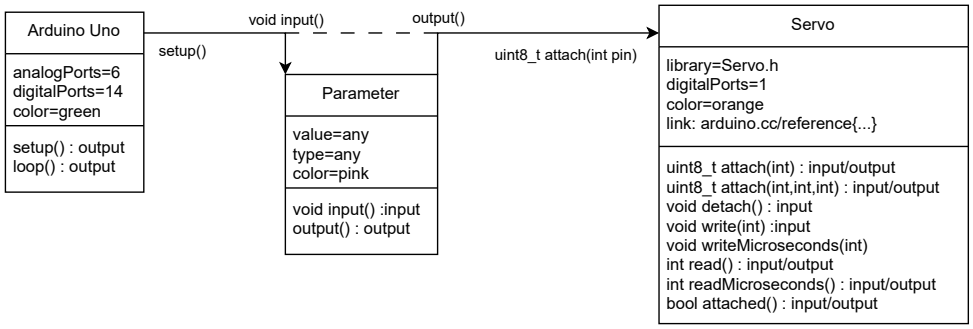


Figure 23 – Link providing single value as parameters.

Multiple values can be chained in order to fulfill methods that require multiple parameters. Value nodes, which include 'Variable', 'Constant', 'Parameter' and 'Port' could have been used interchangeably as parameters in Figures 23 and 24.

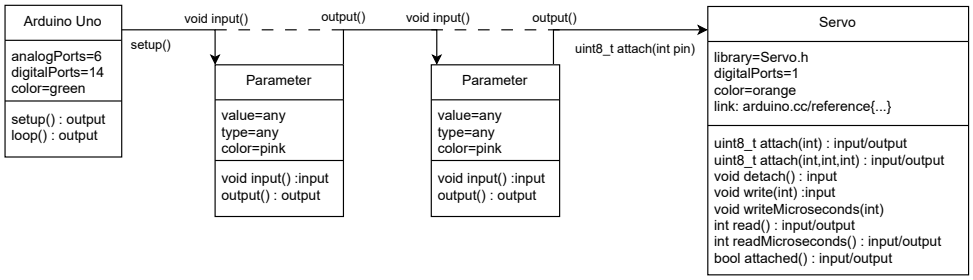


Figure 24 – Link chaining multiple values as parameters.

Additionally, component methods with return values can also be used as a parameter into another method or function, such is shown in Figure 25.

4.1.4 Transformations

The transformation between the user-modeled PSM and executable source code made by the Generator component is depicted in Figure 10. The generator applies a set of specified transformation rules to the diagram until source-code is reached. These rules are currently specified in the tool's source-code, it could be extracted and imported as an external artifact, making it extendable.

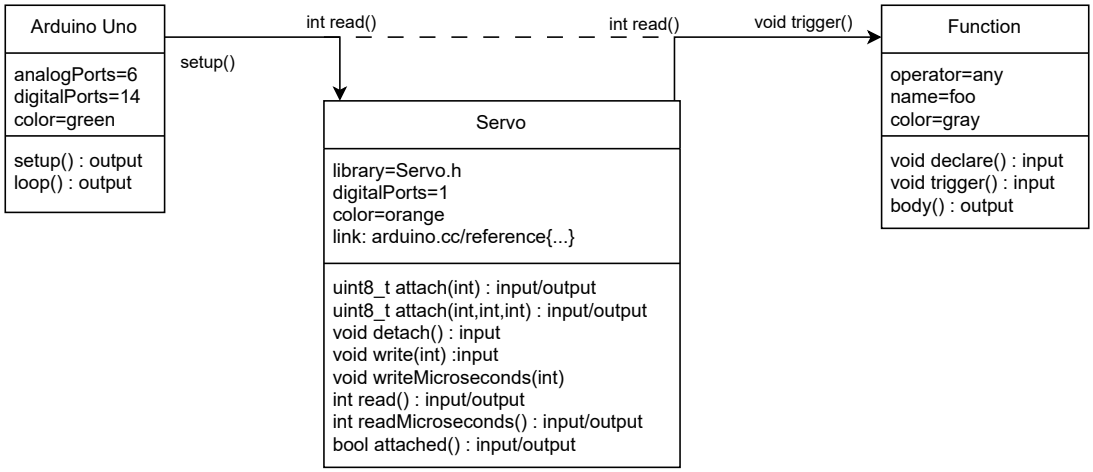


Figure 25 – Link providing method returned value as parameter.

The generator interprets the diagram after any change is made, the interpretation method is very similar to a compiler's parser, (a lexer worker is not needed as the diagram is already divided into nodes, which would be equivalent to the matched tokens in this compiler comparison).

The pseudocode for this required generator is the following: the main routine is shown on Algorithm 1, which will trigger recursively the routine shown on Algorithm 2, additionally, Algorithms 3 and 4 are executed for problem indication.

Data: List of global micro-controllers
Result: Execution of transverse routine
if *number of micro-controllers < 1* **then**
  throw problem 'lacking micro-controller';
end
if *number of micro-controllers > 1* **then**
  throw problem 'too many micro-controllers';
end
for *each micro-controller* **do**
  if *number of links < 1* **then**
    throw problem 'micro-controller not connected';
  end
  execute routine 'transverse';
end

Algorithm 1: Micro-controller connections routine.

```
Data: List of local links
Result: Method triggered or problem thrown
for each link do
    while link.target is value do
        add to local value stack;
    end
    if link.target is not value and is method then
        compare local value stack to method parameter;
        if matched then
            trigger method;
        end
        if not matched quantity then
            throw problem 'unmatched parameter quantity';
        end
        if not matched types then
            throw problem 'unmatched parameter type';
        end
    end
end
```

Algorithm 2: Transverse routine.

```
Data: List of global links
Result: Problems are thrown.
for each link do
    if loose end then
        throw problem 'loose link';
    end
end
```

Algorithm 3: Loose links routine.

```
Data: List of global nodes
Result: Processed global nodes
for each global node do
    if <1 links then
        throw problem 'node not connected'
    end
    if has library then
        if library isn't imported then
            import library
        end
        declare constructed instance
    end
    if constant then
        declare constant on global scope
    end
    if variable then
        if declare function is not linked then
            declare a variable on the global scope
        end
    end
    if function then
        if declare function is not linked then
            declare function on the global scope
        end
    end
    if port then
        if analog then
            add to analog comment
        end
        if digital then
            add to digital comment
        end
    end
end
```

Algorithm 4: Global nodes processing algorithm.

4.2 MDD4CPD Tool

This section describes the development of the tool used to apply the proposed framework. The MDD4CPD tool aims to apply and validate the presented framework, which will be used in both evaluation stages.

The tool has its development correlated to the framework's own development, as result, it has been through multiple iterations, including switching technologies drastically. A current overview of the interface is presented in Figure 26, divided into 3 sections as described below:

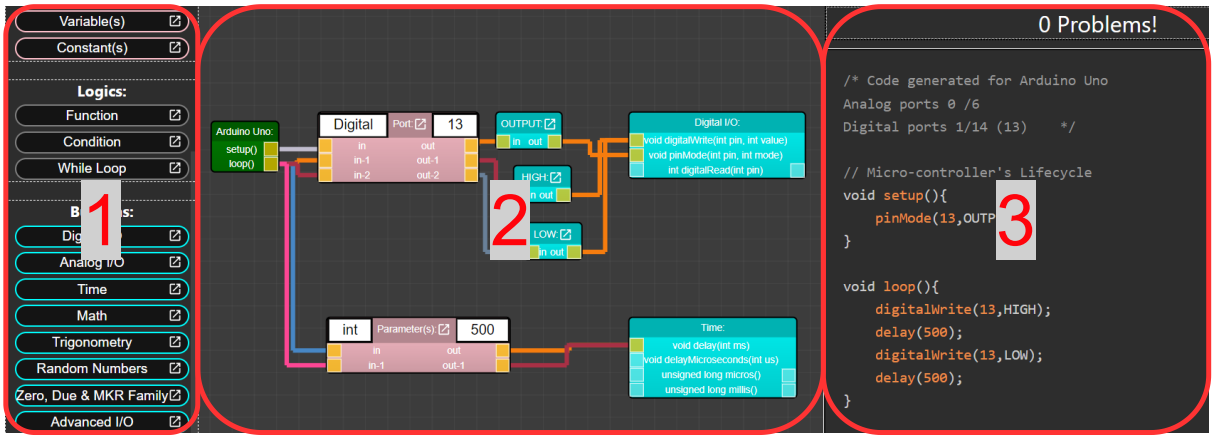


Figure 26 – Example use of the MDD4CPD tool.

1. Palette of components: a list of draggable components, every single one has a corresponding CYM specified in the tool source-code, the components are sorted into the following categories: Micro-Controllers with a collection Arduino models; Value and Logic components as it was specified in the CYM; Built-ins are a collection of non-hardware based components provided in the bundled Arduino libraries, such as the Math or Serial libraries; Built-in-constants are also provided in the Arduino libraries; and finally Components has a small collection of hardware-base components with their respective libraries.

2. The diagram area, where nodes can be dragged and dropped, have their values edited, and their gates connected.

3. The results panel is split into two, firstly a list of problems present in the current diagram, secondly the ready-to-go, auto-formatted, auto-commented, syntax-highlighted source-code. When no problem is found, the source code can be copied and pasted into a simulator or exported into a real device for testing.

The functional software requirements for the tool are brief: the software must allow the user to interact and edit the PSM in order to generate executable source code. It was defined that this model editing was to be done graphically and that the source code would be generated instantly so that the developer could observe the changes made. A C++ linter powers a problem detection system that pragmatically warns users, injecting comments and indicators into the user interface.

The generation of code using the user-edited diagram as the source is done by parsing the diagram, making records of all nodes and links, and additional information not visible in the diagram is also collected, such as the number of ports and libraries used. All the transformation rules are built into the software logic and applied at runtime, it is triggered after every diagram change.

Due to the low processing power needed and high requirement for portability, the tool is made using entirely web technologies. The component-based JavaScript framework React is used, Typescript is used for strong typing, adding robustness to the application. Another essential element is react-diagrams, an open-source package that provides the base diagram interface, which had its source code heavily modified to work on MDD4CPD. The project is also open-source, its source-code and a mostly updated version of the tool are hosted and always available at https://github.com/rzfzr/mdd4.

A custom modeling interface was created, supporting the DSL described in the previous section, understanding as discussed, that the usability of the tool is a key factor for the approach's success, special attention was given to its development.

The tool functions are:

1. The tool parses the static CYM models in order to generate a palette of elements to be used by the user, this palette with elements can be seen on the left side in Figure 26.

2. The tool offers its maintainer a place for inserting the transformation rules, which currently is part of its own source-code.

3. The tool offers the user a PSM diagram modeling interface, in the middle of the screen, where the elements specified in the CYM models can be dragged into.

4. Also inside the diagram interface, the dropped nodes can be connected to others by using links, a connection will have a specific meaning depending on what ports it's connecting.

5. The tool enforces the transformation rules during the live-editing of the PSM diagram, by limiting what connections the user can make and showing error/warning messages in the outputted source-code which is on the right side of the screen.

6. The tool generates executable C++ source-code every time the diagram is updated by the user, the code can then be copied and pasted into an IDE or emulator for execution.

7. The tool offers a model description on hover, with external documentation links, besides the user manual.

4.2.1 Nodes and Links

Recalling the elements present in the DSL representation of the Arduino Uno in Figure 17, its representation inside the MDD4CPD tool is displayed in Figure 27.

The color present before is used as the node's background color, methods with Input will have an Input gate, methods with Output will have output gates, and gates will be the graphical artifact used for attaching links.

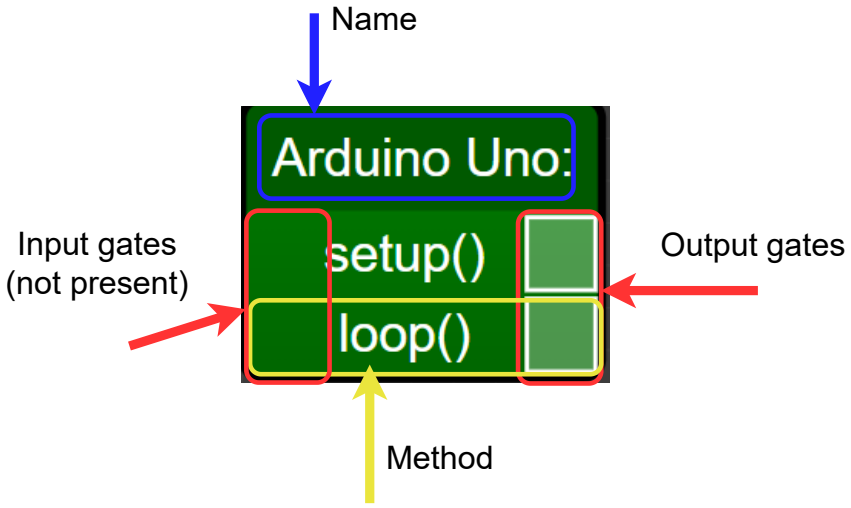


Figure 27 – Description of the Arduino Uno Node.

Figure 28 shows the same component as Figure 18, it has the same GUI elements as the Arduino Uno in Figure 27. These elements include the component's name and a list of methods with their corresponding Input/Output gates.

Logic nodes shown in Figure 29 present two new GUI elements: a text input field used by 'Function' so that the user can name it; and a list selector input used by 'Condition' and 'While Loop' in order pick the desired comparison operator.

Some value nodes shown in Figure 30 such as 'Variable' will have three distinct inputs, a text input for its value; another for its name; and a list selector for its type.

Inside the MDD4CPD tool, links are to be made by connecting an input and an output gate together, by dragging the mouse from one to the other, a parameter-less

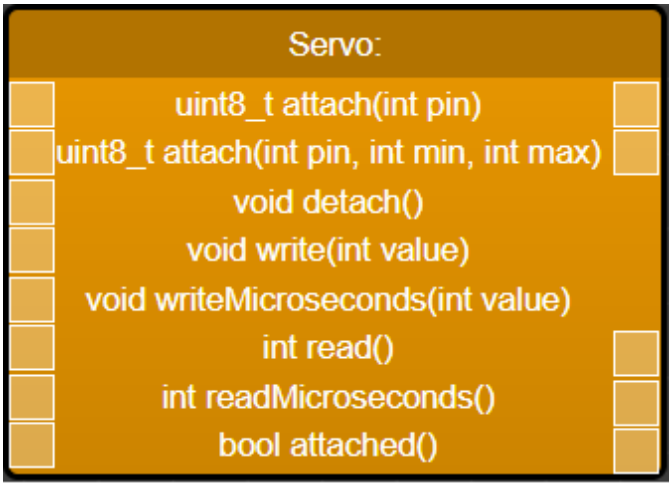


Figure 28 – Servo Node representation in MDD4CPD's PSM.

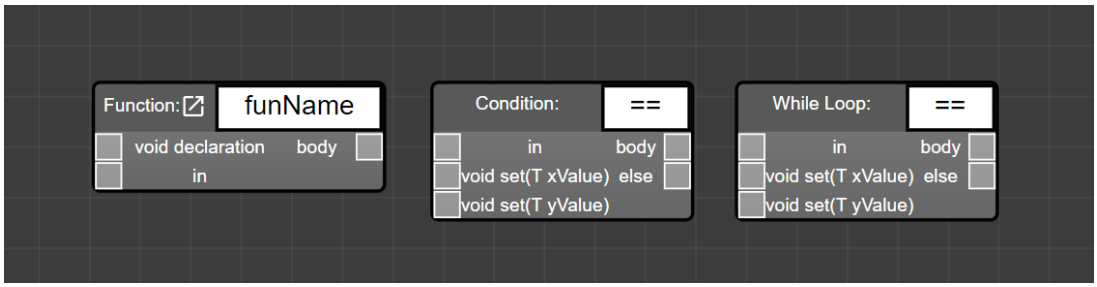


Figure 29 – Custom logic nodes.

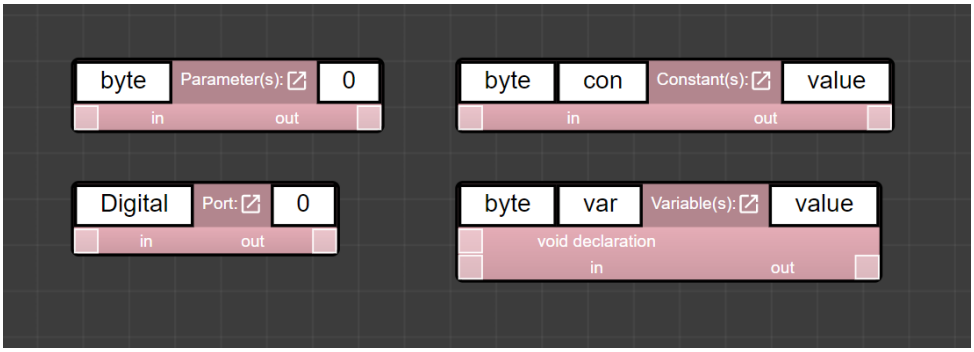


Figure 30 – Custom value nodes.

connection is shown in Figure 31. As the gate is attached to a specific method, not the whole node, the method selection can be explicit without being textual as it is on the DSL in Figure 21.

Implementing the entire DSL specification, links can also be made to trigger a custom 'Function' (Figure 32); carry a parameter (Figure 33); chain multiple parameters (Figure 34); and use component methods with return values as a parameter into another method or function (Figure 35).

The same simple "Uno-Servo" example used throughout the previous section has

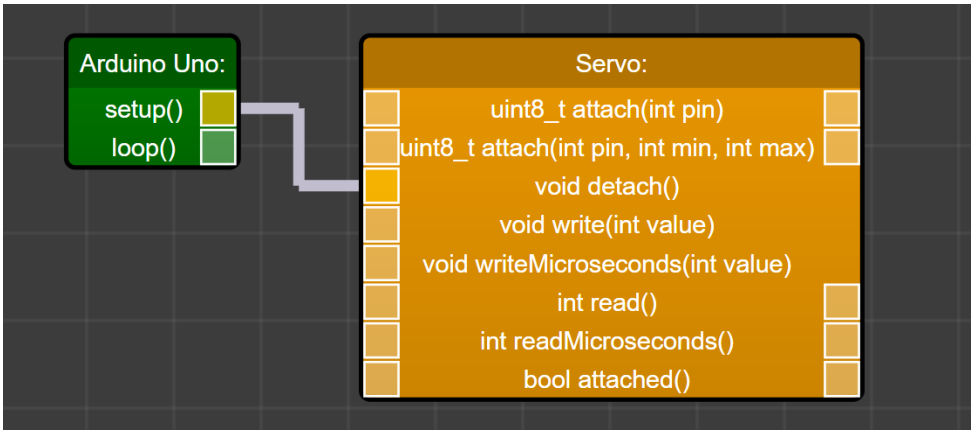


Figure 31 – Link to call the Servo's detach function.

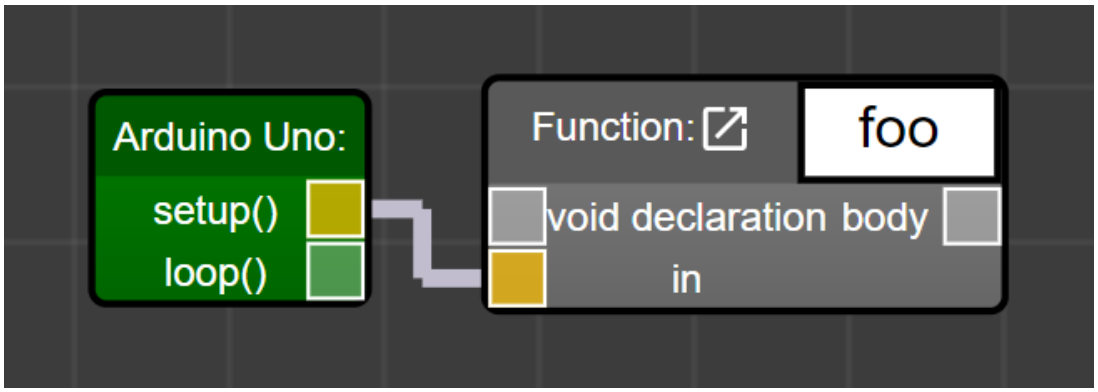


Figure 32 – Link to call the parameterless 'foo' function.

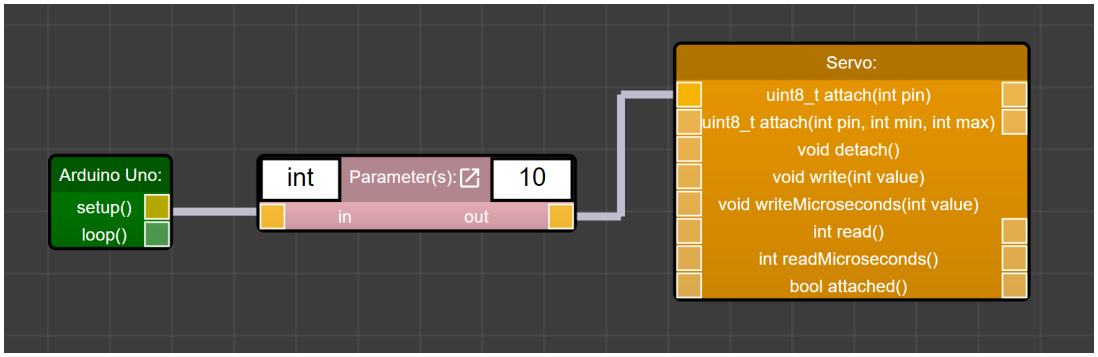


Figure 33 – Link providing single value as parameters.

both static PSM (Figure 14) and dynamic PSM (Figure 14) replaced by the DSL model provided in Figure 16, which inside the MDD4CPD tool is shown in Figure 36.

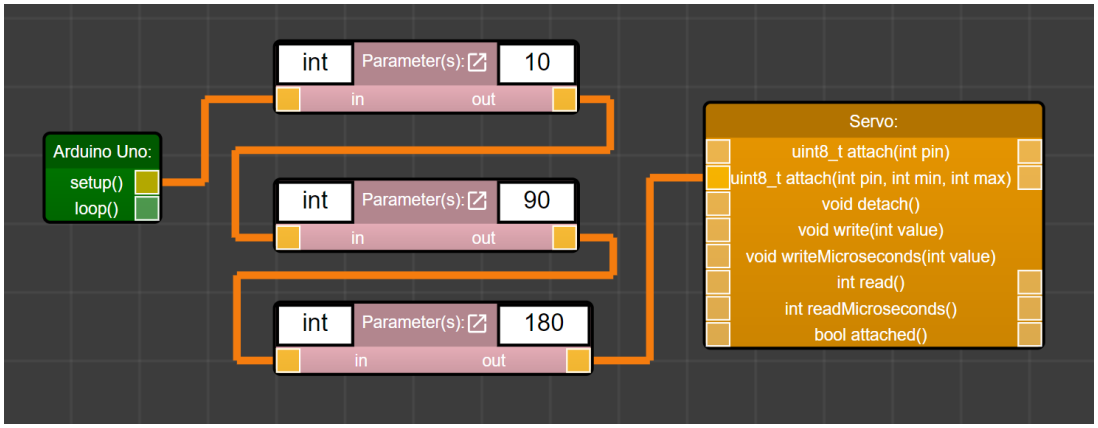


Figure 34 – Link chaining multiple values as parameters.

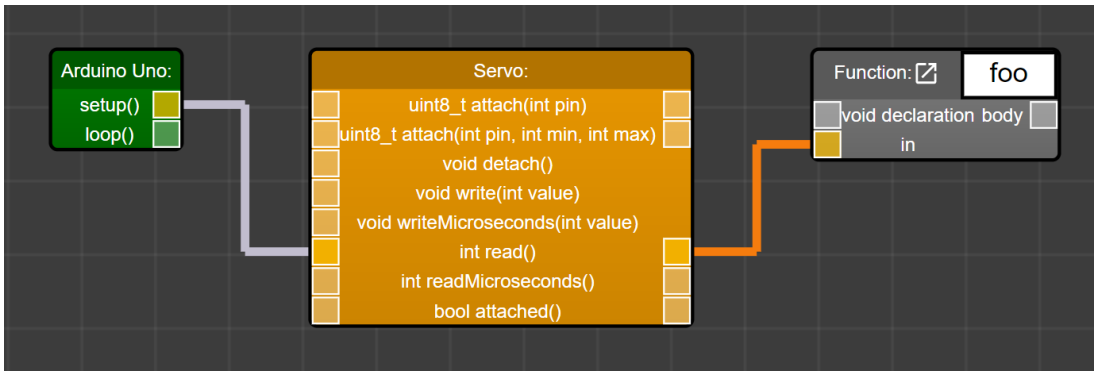


Figure 35 – Link providing method returned value as parameter.

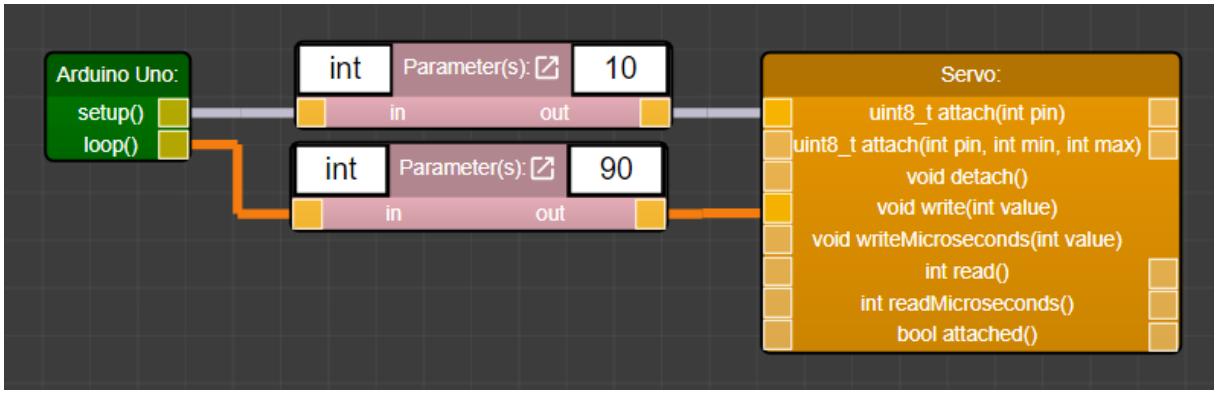


Figure 36 – Complete PSM Modeling inside MDD4CPD.

4.2.2 Additional GUI Elements

In addition to nodes and links, the graphical user interface includes several elements that are not strictly necessary for the tool workflow. These elements were designed to enhance the developer experience and are discussed in detail in this section.

On the component palette shown in Figure 37, each component has an on-hover

icon that displays 3 items: the static CYM which is generated by the tool and used for node creation; a scroll-able panel with a summary of the documentation; and an external link to the official documentation.

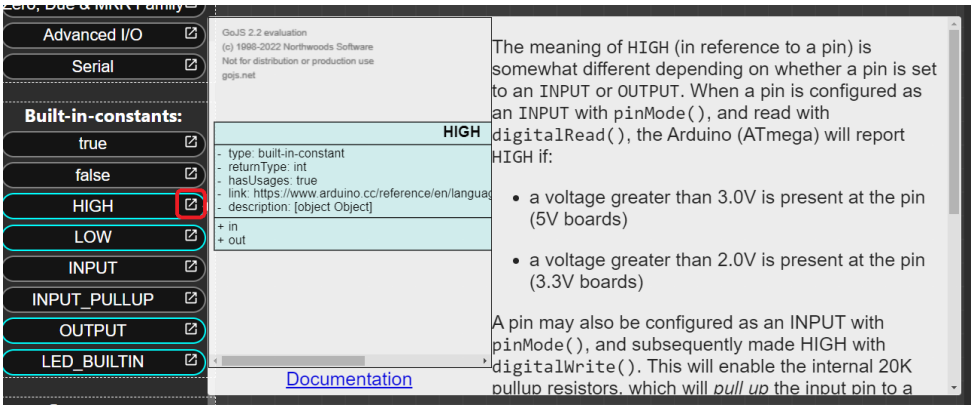


Figure 37 – Additional component documentation on hover.

Value nodes have a special feature demonstrated in Figure 38, a hover icon that displays a popup with the option to add or remove rows of methods, which enables the value to be accessed by multiple components, avoiding duplication.

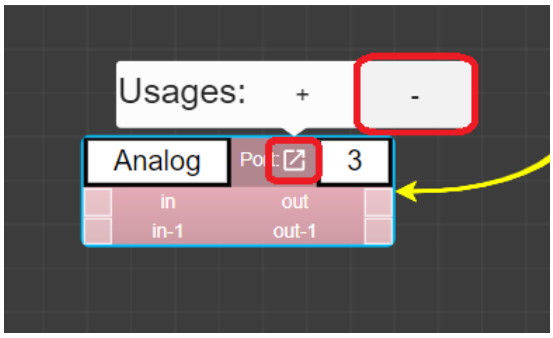


Figure 38 – Additional usages for a specific component.

As we defined the ‘Port’ component being a ‘Parameter’ with syntax sugar, in Figure 39 we can see the advantages: the hidden type=int reduces clutter; the port selector only provides valid values. Furthermore, as an artifact of the generator module, dynamic comments display a port count and listing.

Lastly the MDD4CPD tool has a ‘Problems’ section which is populated by the generator artifacts as shown in Figure 40. It will display a list of warnings/errors as generic problems, all these problems are related to MDD4CPD transformation rules and restrictions, not the C++ language itself. For instance, parameter type mismatch (Figure 41) will not trigger any errors by the C++ compiler, it will be only discovered in execution time.

Figures 41 and 42 show, whenever a problem is related to a specific node, an arrow is generated to guide the use towards the troublesome node.

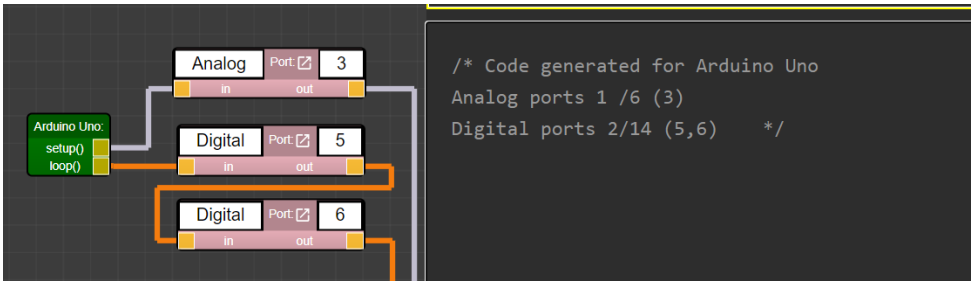


Figure 39 – Dynamic comments listing out micro-controller port usage.

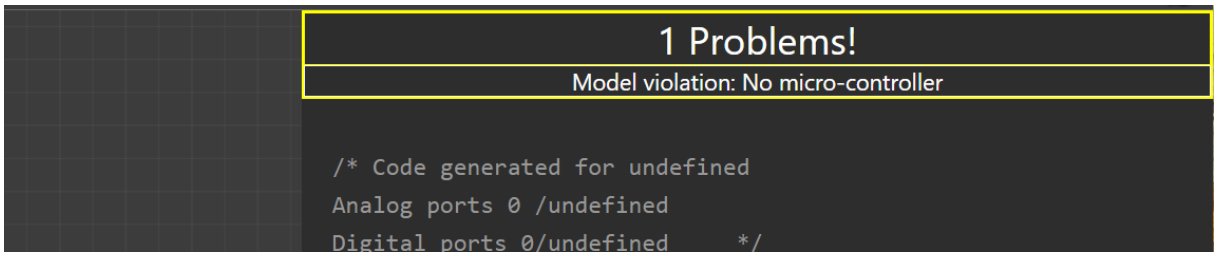


Figure 40 – Lack of micro-controller being pointed out by the ‘Problems’ component.

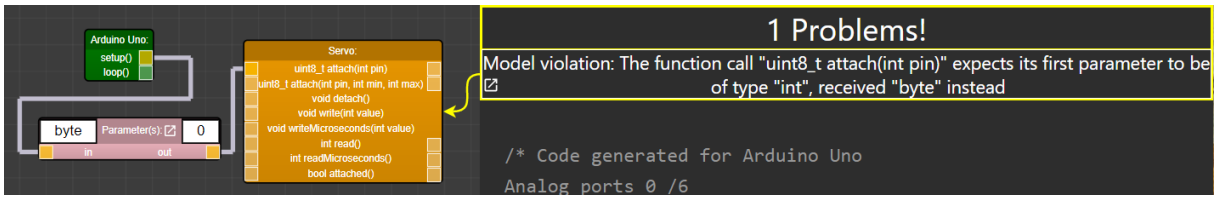


Figure 41 – Parameter type mismatch being pointed out by the ‘Problems’ component.

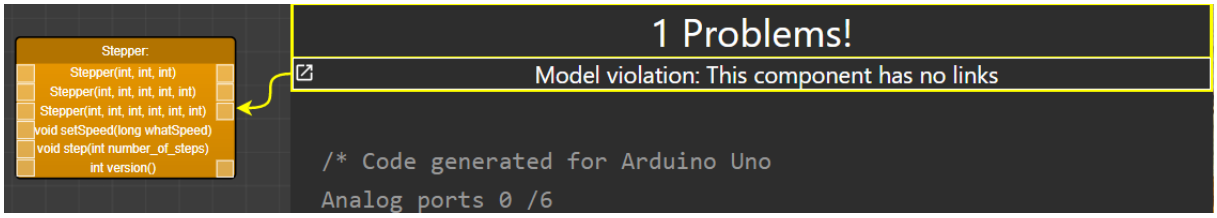


Figure 42 – Lack of link is pointed out by the ‘Problems’ component.

5 INTERNAL EVALUATION

This chapter presents the first step in our framework validation process. It analyses the source-code produced by the framework and tool, in aspects such as completeness, functionality, readability, execution time, and source code quality. This evaluation was previously published [13].

To evaluate our framework's produced solutions, we performed a quasi-experiment by following the guidelines from Wohlin et al. [53]. The authors deemed quasi-experiments as important and capable of providing valuable results, and their definition is the following:

> Quasi-experiment is an empirical inquiry similar to an experiment, where the assignment of treatments to subjects cannot be based on randomization, but emerges from the characteristics of the subjects or objects themselves. [53].

This quasi-experiment aims to assess whether the proposed framework and tool may generate valid Arduino IoT code from the proposed model, and also assess whether the generated code is of quality. For this, we used IoT projects from GitHub [1], and implemented equivalents projects using our tool. In the context of this experiment, valid code refers to code that is executable, error-free, and fulfills its functional requirements.

In order to evaluate a software solution generated with the aid of the MDD4CPD framework, distinct sub-tasks were executed: metrics to be analyzed were selected; a control solution and MDD framework were selected; development of the solution using the framework; the discussion of the results.

5.1 Execution Time

To validate the solution at execution time, a well-established benchmark for Arduinos had to be selected. Testato's SoftwareWire library is available on GitHub since 2008 under the GPL-3.0 licence, it is used for the creation of additional I2C (a serial communication protocol) bus lines on any ports of an Arduino (which usually only has 2 hardware lines, depending on model).

The SoftwareWire library provides a couple of files:
"StressTest_Master.ino" and "StressTest_Slave.ino" which combined form 209 lines of code, with the goal of sending and receiving data through the selected serial lines indefinitely while recording the total time for the communication. This benchmark was

[1] available at <https://github.com>

selected for the comparison between itself and an equivalent solution made with the aid of MDD4CPD.

The equivalent code solution was created by the MDD4CPD's maintainer, using the framework's tool. The original code has been studied and interpreted to be recreated inside the custom modeling language. To note that the solution was only partially automatically generated from the visual representation, the class structure and the object initialization were generated while the decision-making logic was manually written as the tool is not fully capable.

Both original and MDD software solutions were executed in real hardware, to exclude possible bias introduced by simulation. The used Arduinos can be seen in Fig 43

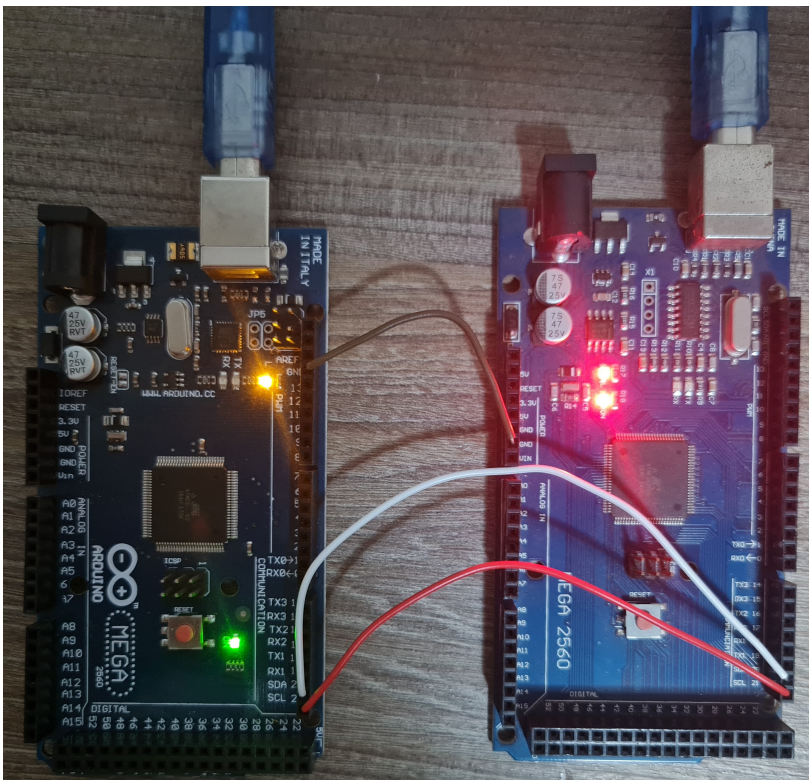


Figure 43 – Image of the Arduinos while connected and running the benchmark.

The benchmark run on the physical hardware to the point of no perceived oscillation in the results, for a total of 55 samples in both the original code (Sample 1) and the MDD4CPD aided solution (Sample 2), the run time results can be seen in Table 1, where both samples obtained equivalent results.

5.2 Software Metrics

To evaluate our approach, we create a simple experiment. This experiment aims to assess whether the proposed approach and tool may generate Arduino IoT code correctly

	Sample 1	Sample 2
Number of Executions	55	55
Mean	248.436	248.345
Median	248	248
Mode	249	248
Range	[247,249]	[247,249]

Table 1 – Data transfer benchmark, execution time results, in milliseconds.

from a model, and at the same time assess the generated code's quality.

We used IoT projects from GitHub as a basis for our study and developed equivalent projects using our tool. The code from GitHub projects is our control and is called Sample 1. The generated code produced by our tool is Sample 2. Afterward, we compared the functionalities of the initial project from GitHub with our version, and also measured and analyzed several quality metrics from both solutions.

We looked for small projects made for Arduino, that included a least one external component, were selected from open projects on GitHub, a widely used code-sharing/ version control platform. We are working under the assumption that code repositories with high popularity are better representing real-world solutions. We used three different projects:

- The first project is an evolution of "blink" (usually Arduino's "hello world" project about a flashing LED) with multiple LEDs flashing in series.
- The second project has three potentiometers (variable resistors) as input devices which control three respective servo motors as output, basically mimicking the input.
- The third and last project uses an ultrasonic sensor in a breakout board, connected to the Arduino. The data from the sensor represents a distance that is read and used to output as 3 LEDs, it is also displayed in the serial monitor, this project can be seen on figure 44.

Three equivalent code solutions were created by the MDD4CPD's maintainer using the tool's assistance. To note that the solutions were only partially automatically generated from the visual representation, the class structure and the object initialization were generated while the decision-making logic was manually written as the tool is not fully capable yet due to time constraints.

All code solutions were tested on TinkerCAD [2], a platform for simulating schematics and Arduino projects which uses the same compiler as the official Arduino IDE. We

[2] available at <https://tinkercad.com>

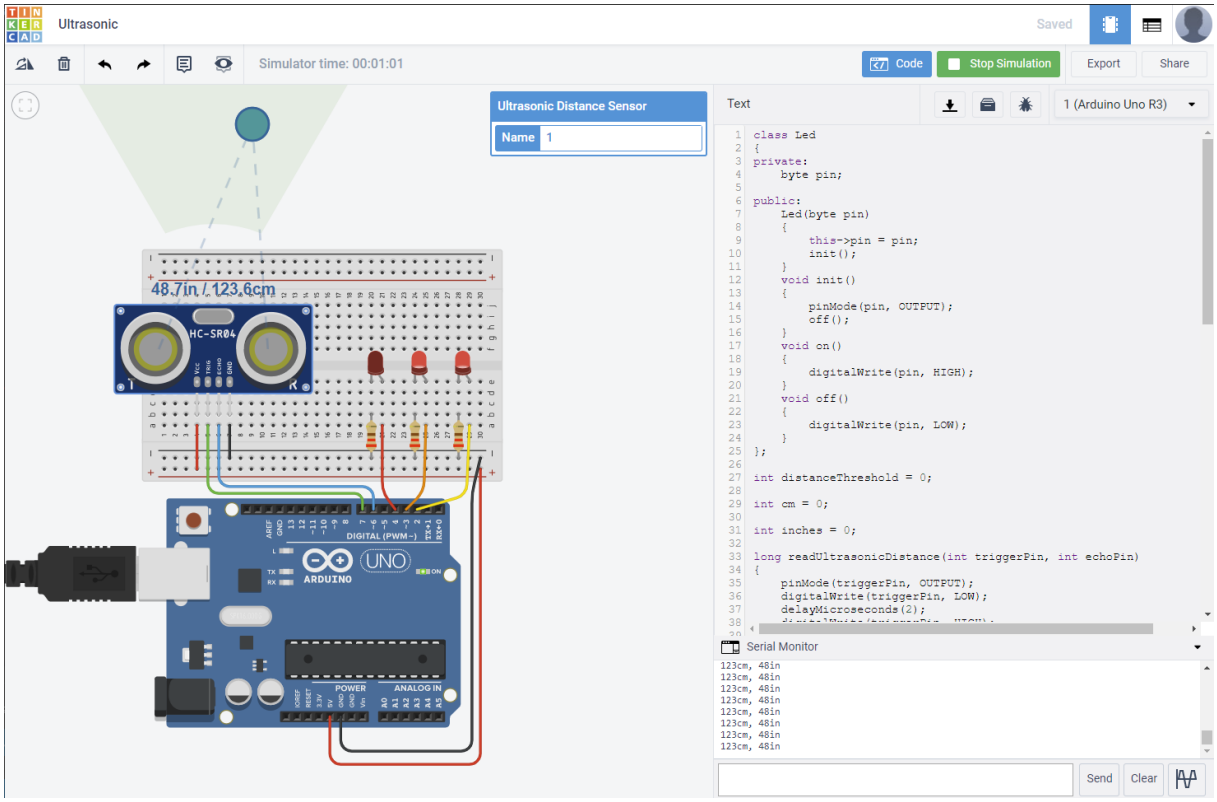


Figure 44 – Tinkercad' simulator executing a generated project.

validated that all three pairs of solutions were working with no errors and the behavior between the pairs where identical, as it should be.

In order to validate the efficacy of the approach, the generated source-code was executed, as a second step the software quality was compared to a conventional solution.

With all this context in mind, we used Nuñez-Varela et al. [54]'s systematic mapping of programming paradigm to software metrics, and we extracted the top ten most popular metrics used for Object Oriented Programming. Table 4 has the explanation for every metric used in the analysis, the descriptions were taken from Analizo's extensive documentation, which has papers supporting each metric.

All metrics were calculated by Analizo, a open-source command-line tool developed by Terceiro et al. [3] and executed in a Xubuntu 18.04 machine. In Figure 45 we can see the result's average of the three projects, higher numbers are worse, meaning higher complexity (Sample 1 = collected code, Sample 2 = generated using the approach).

We separated the metrics into 3 main categories: Size, Coupling and Cohesion, and Other. Analyzing the results by category we can see check for future improvements brought by the use of the framework.

Size has the 2 metrics with the highest delta between samples: LOC and NOA,

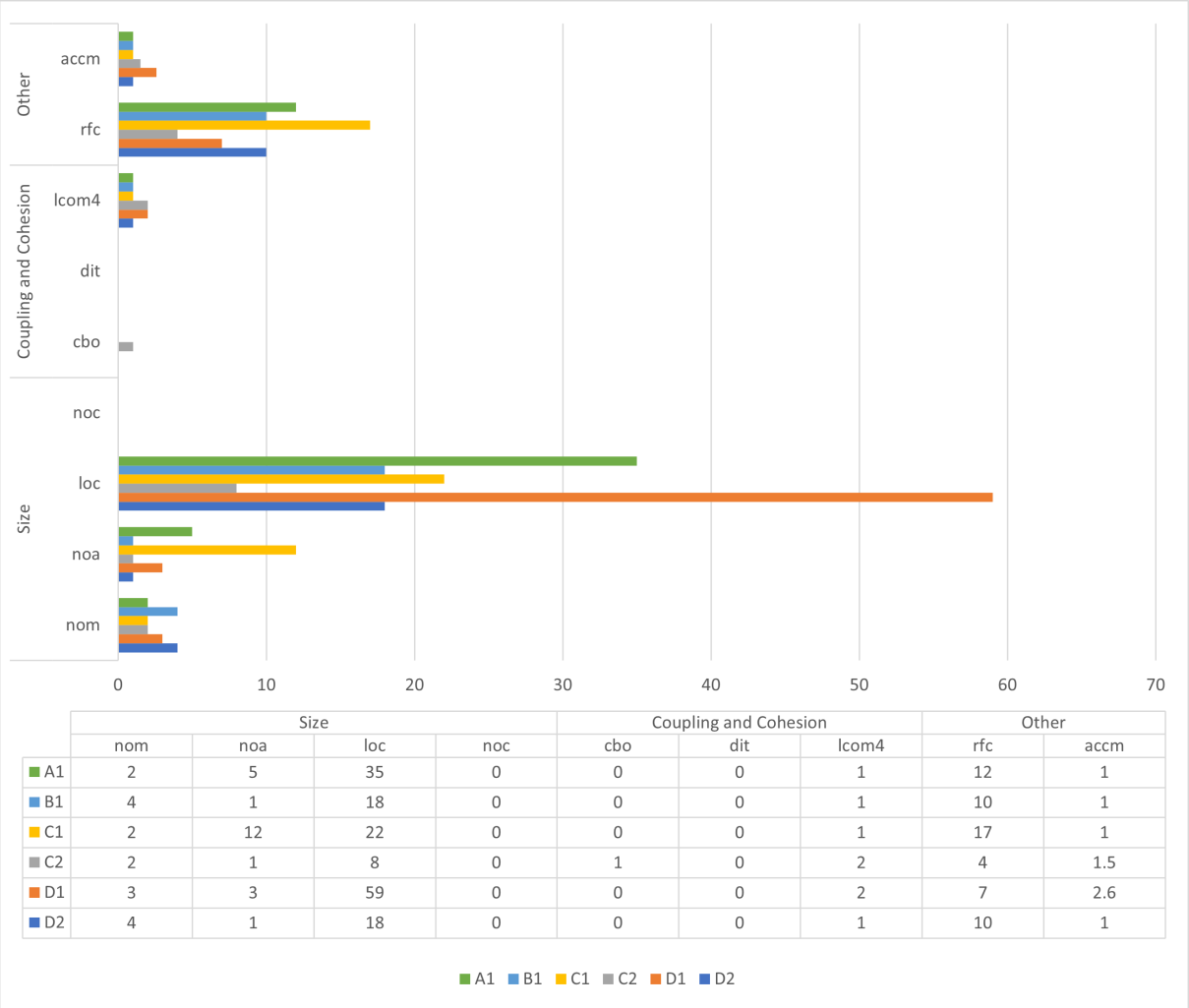
Metric	Description
Coupling Between Objects (CBO)	CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use method or instance variables of another. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.
Lack of Cohesion of Methods (LCOM4)	The LCOM4 value for a module is the number of connected components of an undirected graph, where the nodes are the module's subroutines (methods, functions, etc.), and the edges indicate that two subroutines use at least one attribute/variable in common, or that one subroutine calls the other. These connected components represent independent parts of a module, and modules that have more than one of them have independent, distinct responsibilities.
Depth of Inheritance Tree (DIT)	The metric calculates the longest path from a module to the class hierarchy root.
Lines of Code (LOC) metric	The metric calculates the program size in lines of code, excluding blank lines and comments.
Number of Children (NOC)	The metric calculates the number of immediate sub-classes subordinate to a class in the hierarchy.
Response for Class (RFC)	The metric calculates the sum between the number of methods in the module and the number of functions called by each module function.
Number of Methods (NOM)	The metric calculates the number of methods per class to measure the size of the classes in terms of their implemented operations.
Average Cyclomatic Complexity per Method (ACCM)	The cyclomatic complexity is used in order to calculate the complexity of the program, which can be calculated using a formula of graph theory: v(G) = e - n + 2 where e is the number of edges and n is the number of nodes of the graph.
Number of Attributes (NOA)	Calculates the number of attributes of a class. Its minimum value is zero and there is no upper limit to its result. A class with many attributes may indicate that it has many responsibilities and presents a low cohesion, i.e., is probably dealing with several different subjects.

Table 2 – Software metrics description provided by Terceiro et al. [3].

where Sample 2 is less than half of Sample 1, showing a drastic advantage to the generated solutions. These differences help the developer in maintaining the code base, since it's less than half its actual size. NOM is slightly worse on the generated Sample 2, which is explained by the fact that the less Object Oriented code intrinsically has fewer methods.

Coupling and Cohesion have the least delta in all categories, we suspect to see better results from bigger projects, as there isn't cross-referencing in the samples used.

Other includes RFC and ACCM, both with positive results, which is a consequence of the better code re-use of object instances and their methods. The generated Sample 2 has had overall better results in the metrics used for comparison.



	Size				Coupling and Cohesion			Other	
	nom	noa	loc	noc	cbo	dit	lcom4	rfc	accm
A1	2	5	35	0	0	0	1	12	1
B1	4	1	18	0	0	0	1	10	1
C1	2	12	22	0	0	0	1	17	1
C2	2	1	8	0	1	0	2	4	1.5
D1	3	3	59	0	0	0	2	7	2.6
D2	4	1	18	0	0	0	1	10	1

Figure 45 – Software metrics results between Sample 1 (collected) and Sample 2 (generated) Code.

5.3 Threats to Validity

The obtained results are still preliminary, due to the limited context of models used and the project sample size. Each project's code base size is also not sufficient for a conclusive answer on whether or not MDD4CPD is a viable framework for IoT solutions. Further research is needed, with a more extensive number of metrics and a consolidated selection of quality characteristics.

Our results imply an understanding of the context of the projects used in the evaluation. They were all very basic and small projects, used for instance to flashlights according to a set timer, with input from an ultrasonic sensor (already analog-digital converted off-board).

Secondly, we must understand that the control projects were written in C++, but it has not been designed for taking advantage of Object-Oriented Programming. This is

observed in the metric results, which would differ according to different project scopes and sizes.

There is also a discussion to be had on whether or not all these software metrics are relevant to Internet of Things implementations. As we may find out that certain metrics may have no impact on run time performance or code maintainability. Furthermore, the value proposition of the MDD4CPD implementation could also be discussed, as the initial work of developing the models has a cost.

Each project's code base size is not sufficient for a conclusive answer on whether or not MDD4CPD is a viable framework for large IoT solutions. These results do confirm the main observation, which finds the available code in the GitHub platform is lacking in quality, reuse, comprehension, and maintainability concerns. The bettering of all these properties can be targeted with the proposed approach. Further research is needed, with an improved version of the tool, capable of generating more complex solutions for future comparisons.

5.4 Discussion

Our results from the internal evaluation stage are rather positive, however, we must discuss the meaning of this, with its benefits and limitations.

The execution time results point to equivalent execution speed between code bases, which was expected due to the equivalence between them. This result will scale with project size, provided that the models originate from the same libraries that were used.

Interpreting the software metrics results, firstly we must understand the context of the project used in the evaluation. Secondly, we must understand that the control project was written in C++, but it has not been designed for taking advantage of Object-Oriented Programming. This is observed in the metric results, which would differ according to different project scopes and sizes.

On other hand, the generated code using the tool is Object Oriented. It is the reason why the metrics results we are actually comparing non-Object-Oriented as the control sample to Object-Oriented code generated by the tool. Object-Oriented does have its own overhead, as exemplified by the number of modules, where the generated code had consistently "worse" results (as a higher number of methods signifies higher complexity). These metrics are definitely going to have an inverse result on a larger code base, while some metrics already show better results although the small project size.

Basing this discussion on Klima et al. [38]'s findings, we can classify our results in contrast to the ISO25010 [1]'s quality characteristics.

- Maintainability characteristics such as Analyzability, Changeability, Stability, and

Testability: are potentially impacted by code size, redundancy, complexity, and coupling, this is where the solutions tested differ the most. All relevant metrics extracted, such as Lines of code, code duplication, Cyclomatic Complexity, and Coupling between Objects, were substantially better in the MDD4CPD-aided solution.

- Functional suitability characteristics: are potentially impacted by Unit Test Coverage, something was are not evaluating at this time, between other properties shared by maintainability which had positive results.
- Security characteristics: are possibly impacted by code security and code heterogeneity, which is something that is not being affected by the MDD4CPD-aided development.

ISO25010 [1]'s quality characteristics such as usability that is explored in the next chapter by applying a Technology Acceptance Model. Other quality characteristics such as Portability and Compatibility are deemed not relevant for comparisons, for being left unchanged in both solutions, since the MDD4CPD framework used uses the original micro-controllers libraries and compiles down to the same executable source-code.

6 EXTERNAL EVALUATION

Concluding with the overall positive results throughout our internal validation, described in Chapter 5, which validated software execution and quality. Many more tests can be carried out to evaluate another factor of the MDD4CPD framework. This chapter describes the use of the Technology Acceptance Model (TAM) and the Goal/Question/Metric (GQM), with 14 participants, to judge the usability-related characteristics. These characteristics must be considered since the tool's complexity has been shown to be a prevalent and significant hindrance in previous approaches [22].

TAM is a widely used model for evaluating user acceptance of new technologies. It was first introduced in the 1980s by Davis [55], and it consists of two main factors: perceived usefulness and perceived ease of use. These two factors have a significant impact on the user's intention to use new technology. The evaluation was planned using the GQM system [56], which consists of four steps: Planning, Definition, Data Collection, and Interpretation.

The participants were given a practical user manual, without references to the underlying framework. The user manual described the tool's graphical interface, what were the node properties, and how to link them. Late, the participants were given a video walk-through of the development of the 'blink' project using the MDD4CPD tool, after watching the videos they were instructed to replicate the solution, if possible by memory, if not by referencing the provided material. The participants were asked to fill out two questionnaires, one before and one after the experiment.

Besides the questionnaires, the experiment was recorded and analyzed with the object of extracting data through observations, which is a valid investigation technique as per Wohlin et al. [53]:

> Observations can be conducted in order to investigate how software engineers conduct certain tasks. This is a first or second-degree method according to the classification above. There are many different approaches for observation. One approach is to monitor a group of software engineers with a video recorder and later on analyze the recording. [53].

6.1 Planning and Definition

The GQM model is presented in Figure 46, where twenty-six questions (Table 3) and 5 metrics (Table 4) were developed across two different questionnaires to be applied with a controlled experiment. The two questionnaires are described below:

- Questionnaire 1 is applied before the controlled experiment, it is composed of questions 1-14, besides targeting goal 1, this questionnaire also has additional questions (1-7) which are used for participant profiling and have not been included in the GQM model.
  - Goal 1 is set to evaluate the extent to which participants value specific development artifacts, such as practices or paradigms.
- Questionnaire 2 is applied after the controlled experiment, it is composed of questions 15-26, and it targets goals 2, 3, and 4.
  - Goal 2 is set to evaluate the tool's perceived ease of use.
  - Goal 3 is set to evaluate the tool's perceived ease of usefulness.
  - Goal 4 is set to evaluate the tool's provided documentation role.

The QGM as related to TAM was inspired by the study presented in [57] and the questions were evaluated according to the Likert scale McIver & Carmines [58]. Both questionnaires contained blank fields for comments.

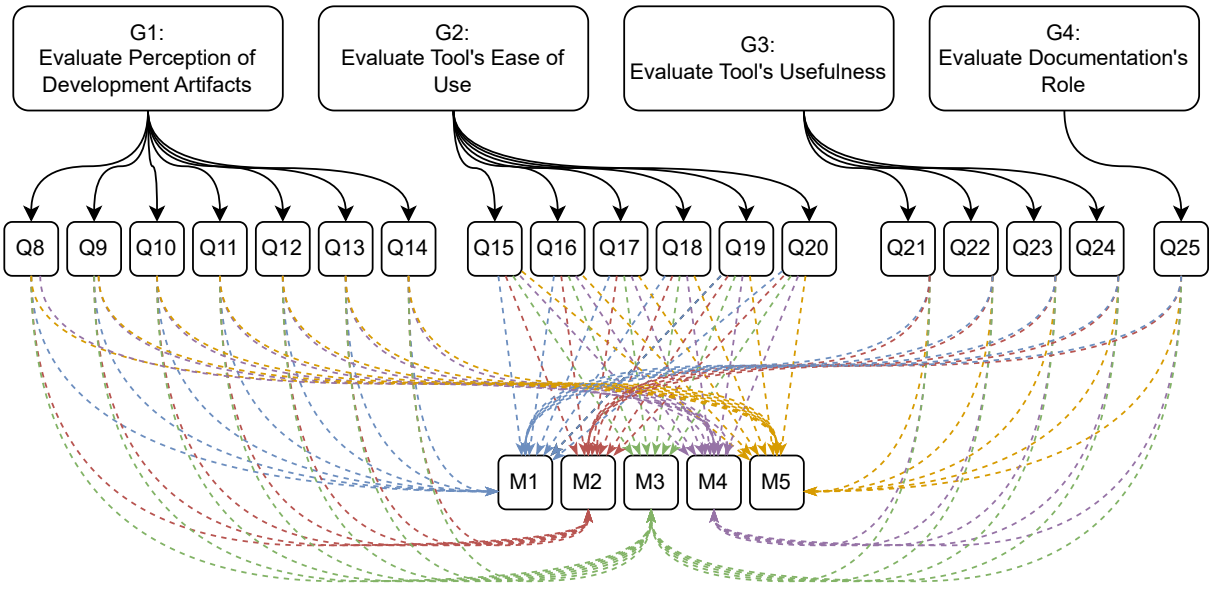


Figure 46 – The developed GQM model.

Question	Description
Q1	Field of Education
Q2	Level of Education
Q3	Professional Profile
Q4	Experience with Object-Oriented Development
Q5	Experience with Cyber-Physical Systems or Internet of Things Development
Q6	If you answered differently from “none” in question 5, what is your experience with cyber-physical systems or Internet of Things?
Q7	Any relevant comments you would like to make about your profile.
Q8	The use of Diagrammatic Models artifacts in software development is important.
Q9	Business rules restrictions provided by a model are important artifacts in software development.
Q10	Object-Oriented Programming is an important paradigm in software development.
Q11	Code organization is an important factor in software development.
Q12	An environment with error detection is an important factor in software development.
Q13	An environment with easily accessible documentation describing the methods and attributes of the components used is an important factor in software development.
Q14	Code generation time is an important factor in software development.
Q15	In general, it was easy to learn how to use the tool.
Q16	I had ease in manipulating the diagram.
Q17	I understood how the process of creating node components was done.
Q18	I understood the interaction between the node components, performed by links.
Q19	The errors and problems pointed out by the tool aided the development process.
Q20	I succeeded in generating the “blink” example solution.
Q21	The use of the tool helps to generate the solution.
Q22	The use of the tool reduces the need for domain knowledge.
Q23	The use of the tool aids in error detection and prevention.
Q24	The use of the tool aids in code organization.
Q25	Documentation was an important item for using the tool.
Q26	Any comment you want to make about the tool.

Table 3 – Questions used in the GQM model.

Metric	Description
M1	Number of people who chose “disagree”
M2	Number of people who chose “slightly disagree”
M3	Number of people who chose “neither agree nor disagree”
M4	Number of people who chose “slightly agree”
M5	Number of people who chose “agree”

Table 4 – Metrics used in the GQM model.

6.2 Demographics

To carry out the evaluation, a group of 14 professional developers was selected to participate in the experiment. They were invited by e-mail (present in Appendix B). Figures 47, 48, 49 show general information about the participants' profiles. To note the high number (92%) of higher education, most (78%) have a degree higher than Bachelor's. They also work across a diverse range of academic and industry fields.

Figure 47 – Participants' education level.

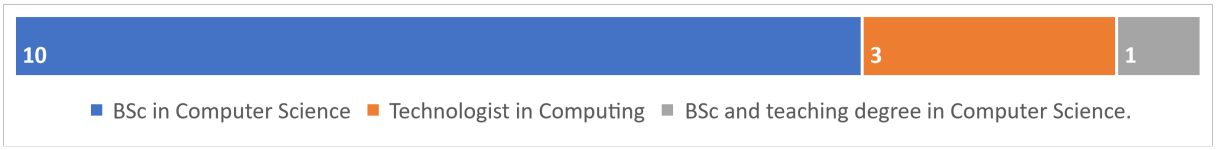


Figure 48 – Participants' professional field.

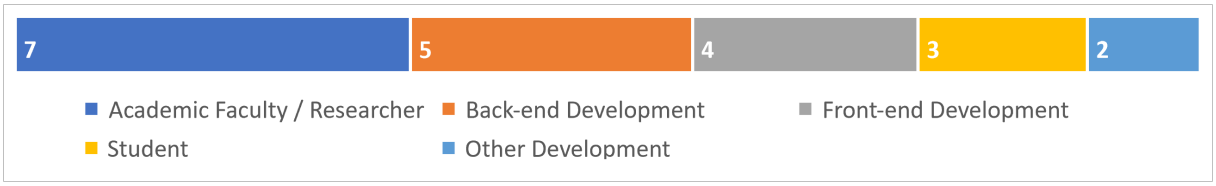


Figure 49 – Participants' occupation.

Figures 50, 51, 52, 53 show data about participants' experience level in aspects with regards to CPD development. Only 14% of participants had less than 3 years of experience in Object-Oriented programming. Most (92%) had prior experience with CPD development, but only 15% of them had experience with platforms besides Arduino or Raspberry. Most of the participants' prior projects (92%) were of academic or personal complexity.

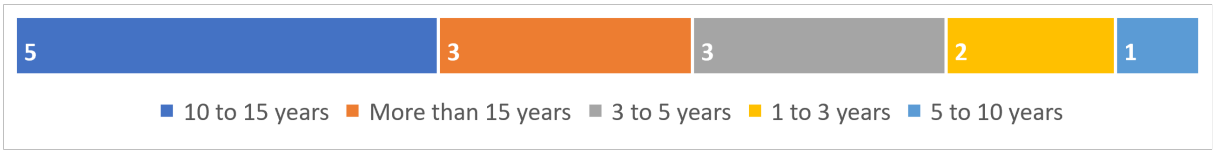


Figure 50 – Participants' experience time with Object-Oriented programming.

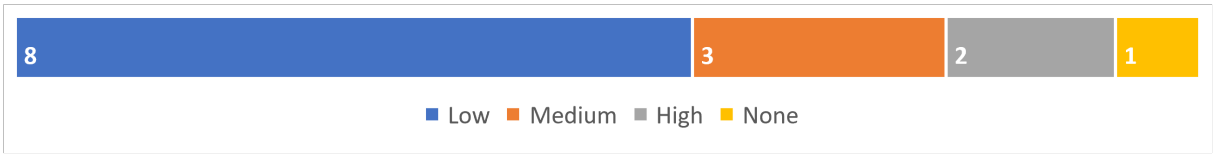


Figure 51 – Participants' CPD development complexity.

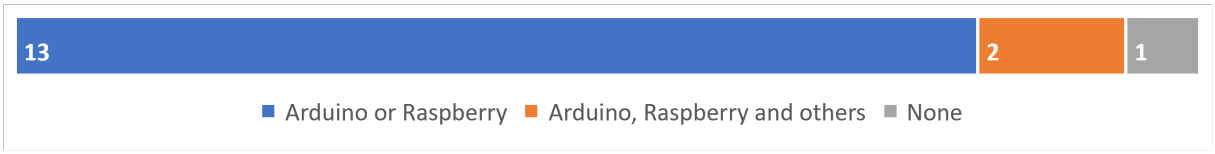


Figure 52 – Participants' CPD platform usage.

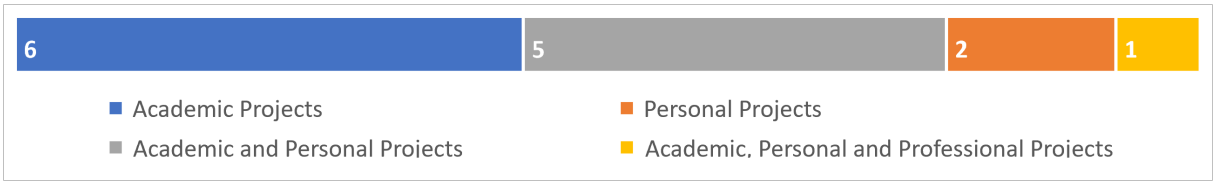


Figure 53 – Participants' CPD development context.

6.3 Data Collected

Figures 54 and 55 show the data collected by Questionnaire1, for questions 8 through 14, which targeted Goal1 in our GQM model shown in Figure 46 (questions 1 through 7 consist of profile data, not included in the GQM).

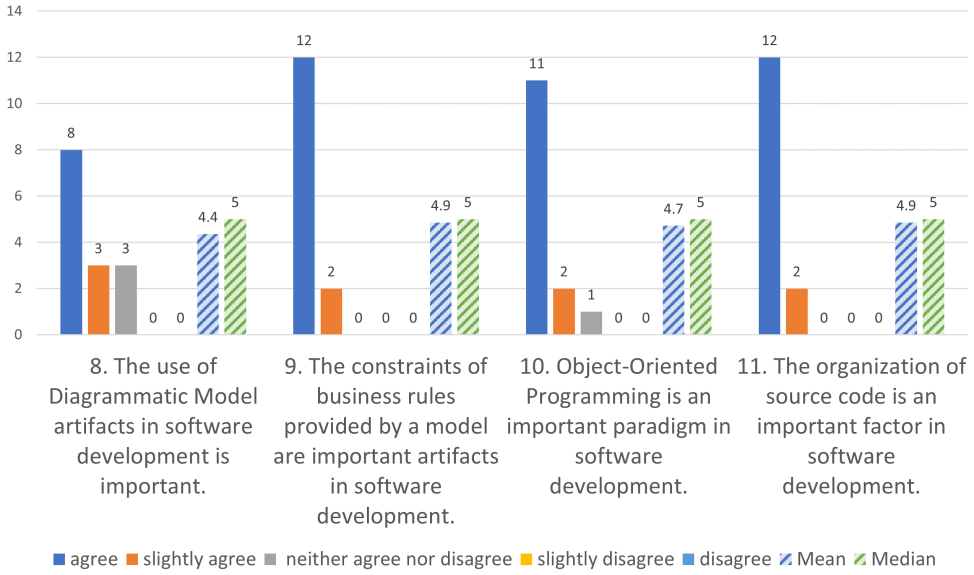


Figure 54 – Questions 8 through 11, Questionnaire1.

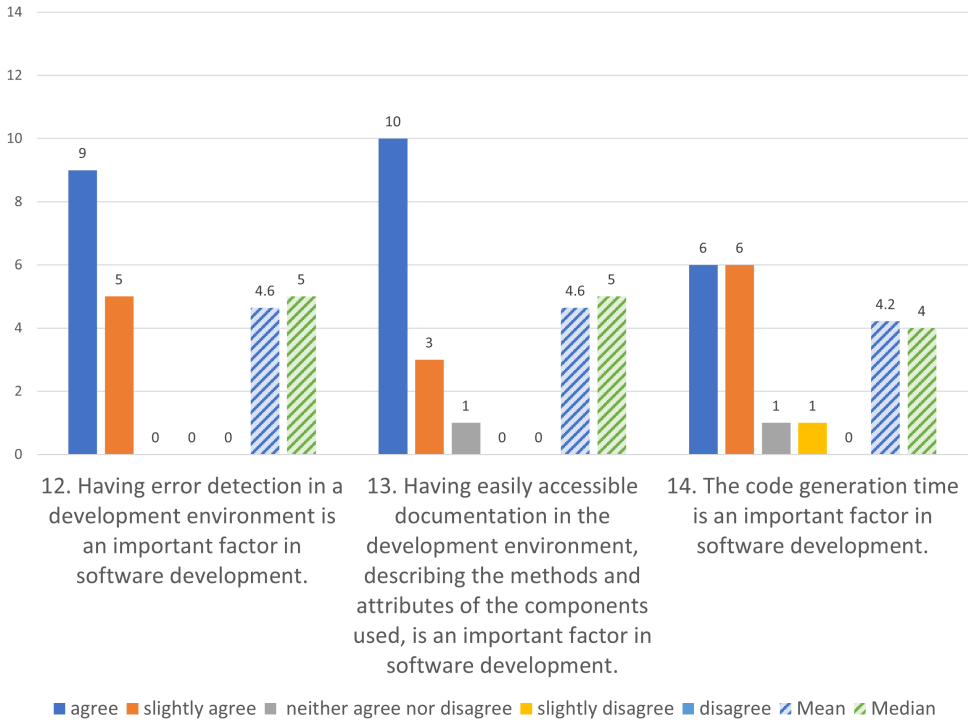


Figure 55 – Questions 12 through 14, Questionnaire1.

The remaining questions were answered on Questionnaire2, Figures 56 and 57 shows questions 15 through 20, which target Goal2; Figure 58 shows questions 21 though 24, which targets Goal 3; and Figure 59 shows question 25, which targets Goal 4.

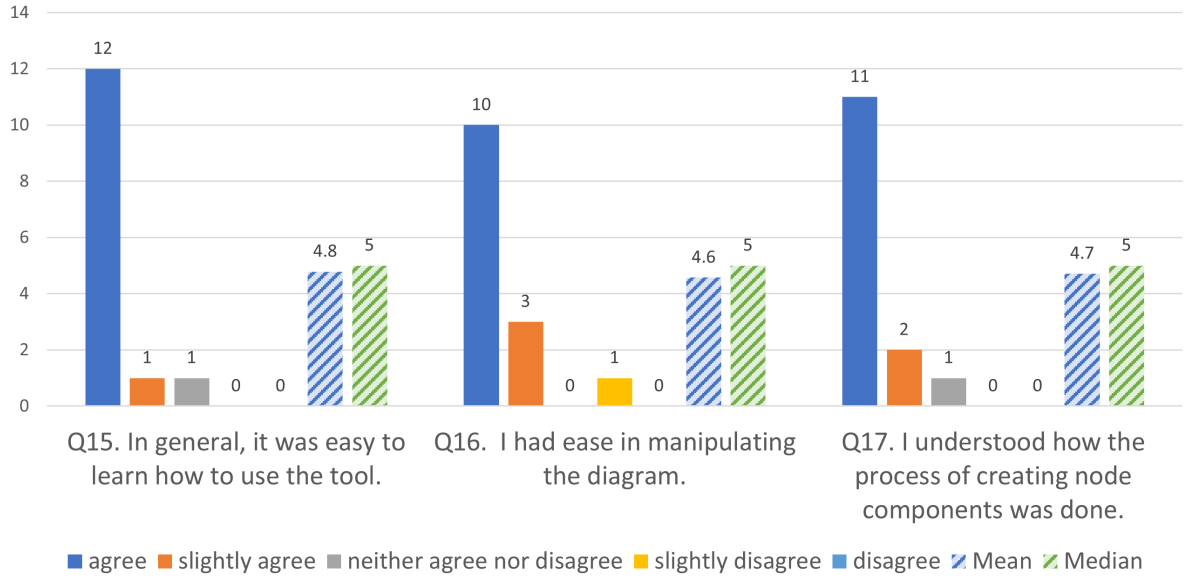


Figure 56 – Questions 15 through 17.

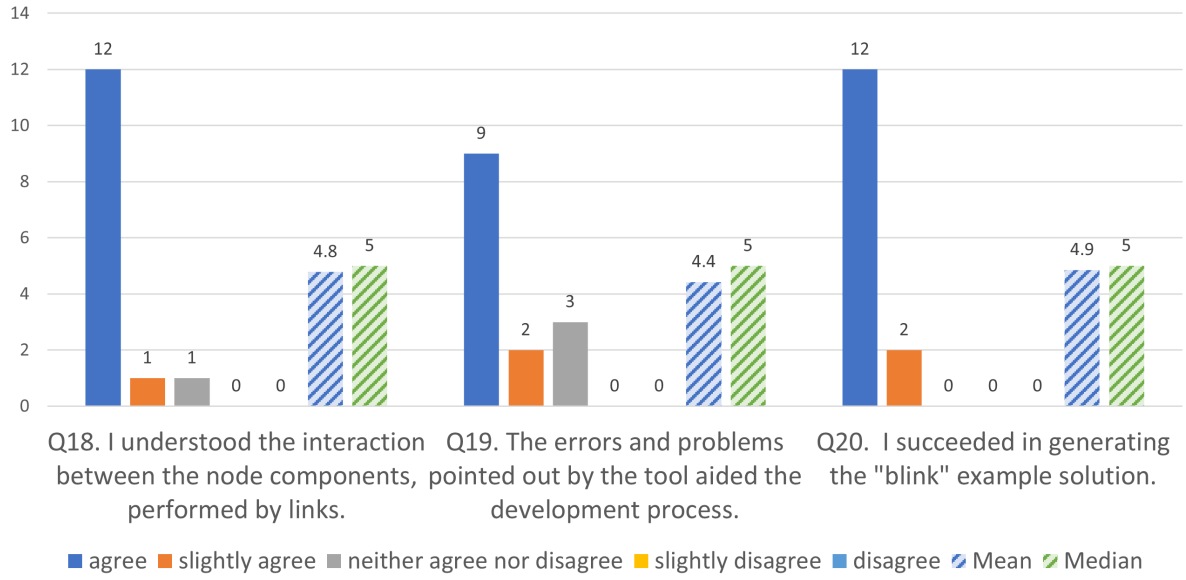


Figure 57 – Questions 18 through 20.

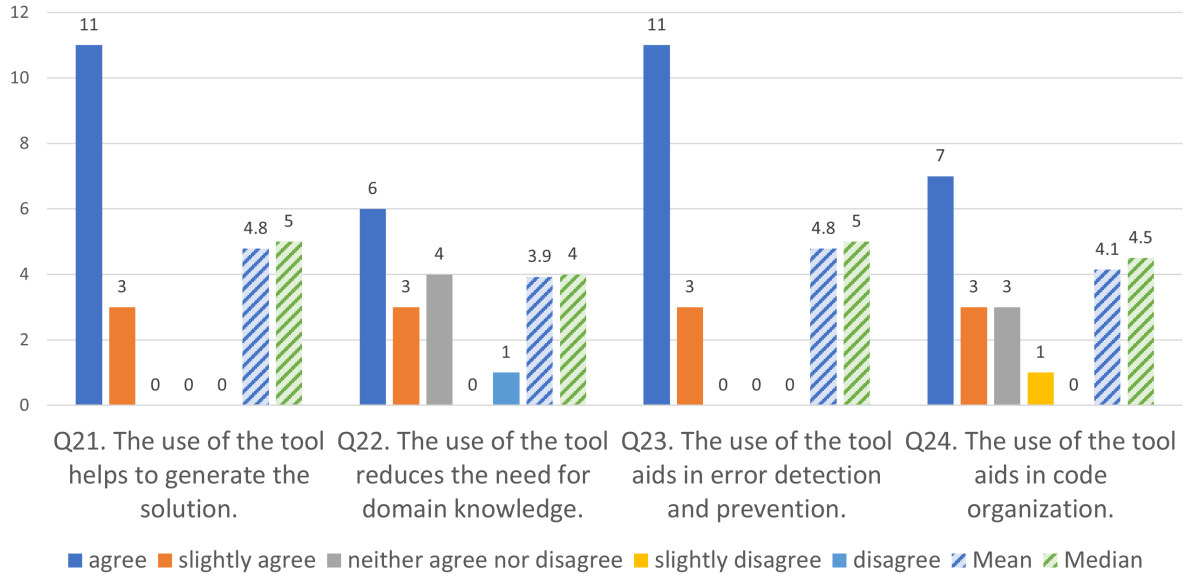


Figure 58 – Questions 21 through 24.

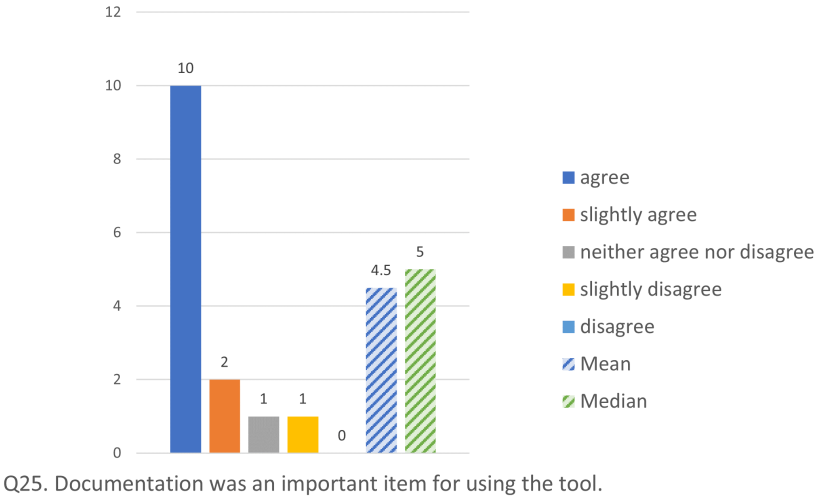


Figure 59 – Question 25.

Table 5 shows all the data gathered from observations, observed items include:

- Manual Time: the amount of time the participant spent reading the provided tool manual.
- Skimmed Reading: whether the participant skipped whole sections of the manual or not.
- Video Interrupted: whether or not the participant had to pause and/or reverse the 'blink' implementation video on the first watch, before the implementation instruction.
- Implementation Time: how long it took for the participant to finish or give up on implementing the 'blink' solution.
- Video References: How many times the user referenced the reference video, after starting to implement the solution.
- Completed: whether or not the participant successfully completed the implementation, the implementation is deemed completed if when executed would produce the asked result, regardless of errors or warnings.
- Deleted Nodes: the number of nodes that were dragged into the diagram from the palette, and then deleted by the participant.
- Rearranged Nodes: whether or not the participant re-located nodes for organizational reasons, without a strict necessity, an example of necessity would be to uncover a hidden gate.
- Critical Errors: the number of errors that would prevent the code from executing correctly, these were counted at the time the participant deemed it finished, or gave up.
- Warnings: the number of warnings that were displayed by the MDD4CPD interface, but which would not prevent the code from executing correctly, such as mismatched types, these were counted at the time the participant deemed it finished, or gave up.
- Total Time: the controlled experiment's entire duration starts from the introduction and ending at the end of the implementation step.

Participant Id	Manual Time	Skimmed Reading	Video Interrupted	Implementation Time	Video References	Completed	Deleted Nodes	Rearranged Nodes	Critical Errors	Warnings	Total Time
1	08:50	No	Yes	23:58	14	Yes	11	Yes	0	0	48:07
2	06:40	No	No	07:00	0	Yes	0	Yes	0	0	47:35
3	06:15	Yes	No	07:40	7	Yes	0	Yes	0	0	41:15
4	07:58	No	No	05:45	8	Yes	0	Yes	0	1	25:15
5	08:40	No	No	04:42	0	Yes	0	Yes	0	0	31:52
6	09:05	No	Yes	11:20	8	Yes	2	Yes	0	0	28:29
7	05:47	No	Yes	10:50	15	Yes	0	Yes	0	0	22:56
8	07:00	No	Yes	09:50	1	Yes	1	Yes	0	0	36:44
9	07:40	Yes	Yes	20:20	8	Yes	0	Yes	0	0	42:32
10	07:18	No	Yes	10:20	16	Yes	0	Yes	0	0	39:40
11	06:10	Yes	No	05:15	0	Yes	0	Yes	0	1	26:10
12	11:00	No	Yes	09:20	5	Yes	1	Yes	0	2	34:12
13	03:40	Yes	Yes	05:00	3	Yes	0	Yes	0	1	16:52
14	08:05	No	No	10:20	10	Yes	1	Yes	0	0	42:11

Table 5 – Participant data.

6.4 Discussion

In this section we interpret the results presented in Section 6.3 in relation to the initial goals outlined in Figure 46:

- Goal1 (Q8-Q14): in relation to the participants' existent conviction about the importance of techniques and practices, we can observe a high consideration for the organization of source code; Object-Oriented programming; and model-provided business rule constraints. A development environment that contains error detection and easily accessible documentation had mixed responses, with a 4.6 mean. Therefore, we can notice a light nonconformity between the perception of importance and the use of documentation in this experiment. The use of Diagrammatic Model artifacts had still an overall positive outcome, but scored lower than others, achieving a relatively low mean of 4.4 in this goal. The aspect deemed less important was code generation, with the lowest mean of 4.2.
- Goal2 (Q15-Q20): in relation to the MDD4CPD's tool's ease of use, the results showed that most of the participants agree with the ease of use of the tool. The participants agreed that: in general, it was easy to learn how to use the tool; they had ease in manipulating the diagram; they understood the process of creating nodes; and the interactions between nodes. The relatively less agreed upon a question, with a 4.4 mean, is in regards to the role of error and problem detection aiding in the development process. All participants conclude the controlled experiment by reproducing the instructed solution.
- Goal3 (Q21-24): in relation to the MDD4CPD's usefulness, the results showed an overall positive agreement on code generation and error detection. The participants had a relatively lower agreement level, with a mean of 4.1 about the use of tools for code organization. The lowest agreed-upon aspect, with a mean of 3.9 was the tool's role in the reduction of required domain knowledge, which maybe was been influenced by the experiment's small scope.
- Goal4 (Q25): In relation to the MDD4CPD's accompanying documentation, it was deemed important for the use, with a 4.5 mean.

Analyzing the blank fields for comments, they were summarized:

- 4 Mixed comments:
  1. Praises ease to use, criticizes visual complexity.
  2. Praises intuitive UI criticizes project scope, the participant feels it would be more adequate on more complex implementations.

3. Praises the component palette interface and criticizes the lack of link order indication.
4. Praises the easy-to-use, criticizes the lack of link order indication.

- 2 Negative comments:

1. Criticizes link complexity.
2. Criticizes the use of web-based technologies.

- 4 Positive comments:

1. Praises all tool features and suggests teaching potential.
2. Praises the use for quick development and suggests teaching potential.
3. , 4 , and 5 all suggest teaching potential.

The results of the evaluation showed an overall positive perception of the MDD4CPD framework's perceived usefulness and perceived ease of use among professional developers. The participants appreciated the tool's graphical interface and found it easy to use, with a high degree of perceived usefulness in their work.

The TAM evaluation provided valuable insights into the MDD4CPD framework's usability-related characteristics, which can be used to improve the tool's design and functionality. Further tests can be carried out to validate the findings of this evaluation and to better evaluate the MDD4CPD framework's potential in software engineering.

7 CONCLUSION

In this study, we introduced, examined, and debated the feasibility of the MDD4CPD framework based on MDD, which is tailored for the development of Cyber-Physical Devices. The framework includes an associated tool and was evaluated through internal and external tests, yielding overall positive outcomes.

7.1 Contributions

MDD has demonstrated itself to be an interesting approach to assist in developing quality and standardized solutions for specific domains. However, its use depends on a specific tool that aids its use. It is also necessary to have high-level models that guide development, a requirement that can be challenging to fulfill in practice.

Low-code development is a trend that has shown to be viable for cyber-physical devices, especially Arduino. Cyber-Physical development, being related to code, hardware, and libraries that integrate code and hardware, was of great importance to have a Low-Code tool that aids this process by facilitating the use of the correct libraries and aiding in the development of solutions. The main contributions of this research are:

- The proposal of an MDD-based framework.
- The creation of a new meta-model CYM, which is specific for CPD devices, it provides not only a high-level model like the MDA's CIM but also integrates libraries and physical devices.
- The implementation of a Low-Code, Open-Source, web-based tool allows the implementation of the approach for CPD code generation. This tool has the following main advantages:
  - Instant and complete code generation, the code is well-formatted, organized, and generated using OOP concepts.
  - A custom and unified PSM modeling stage, fully graphical and model-enforcing.
  - A vast selection of library and hardware-based Arduino components.
  - Custom parser for extensive error and warning generation.
  - Integrated documentation with external links.

7.2 Future Work

Our current work leaves many options for future research and development projects, which can be listed:

- Further internal validation with bigger projects, in order to reinforce the work done in Chapter 5.
- It is evident after the findings in Chapter 6 that some tool's interface changes are required, especially in regard to diagram complexity and link order. After said changes, another controlled experiment could be carried out in order to contrast the results with our first external validation.
- A feature that is absent in all of the related work, but is on MDD4CPD's roadmap is for two-way transformations between PSM and source-code. The tool would be capable of generating diagrams by parsing source-code, making it possible to benefit from the framework's benefits even in a deep software maintenance stage.
- A parser can be easily implemented for analyzing component libraries, and producing CYM's without any manual input.

BIBLIOGRAPHY

[1] ISO25010. *ISO25010 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. [S.l.], 2011. v. 2011.

[2] NGUYEN, X. T. et al. FRASAD: A framework for model-driven IoT Application Development. In: IEEE. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. [S.l.], 2015. p. 387–392.

[3] TERCEIRO, A. et al. Analizo: an extensible multi-language source code analysis and visualization toolkit. In: *Brazilian Conference on Software: Theory and Practice (CBSoft) – Tools*. Salvador-Brazil: [s.n.], 2010.

[4] INTERNETWORLDSTATS. *World Internet Users Statistics and 2019 World Population Stats*. 2019. Disponível em: <https://www.internetworldstats.com/stats.htm>.

[5] HOWELL, J. *Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says*. 2017. Disponível em: <https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says>.

[6] SCHWAB, K.; DAVIS, N. *Aplicando a quarta revolução industrial*. [S.l.]: EDIPRO, 2019.

[7] GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167739X.

[8] VORAKULPIPAT, C. et al. Recent challenges, trends, and concerns related to iot security: An evolutionary study. In: IEEE. *2018 20th International Conference on Advanced Communication Technology (ICACT)*. [S.l.], 2018. p. 405–410.

[9] HAILPERN, B.; TARR, P. Model-driven development: The good, the bad, and the ugly. *IBM systems journal*, IBM, v. 45, n. 3, p. 451–461, 2006.

[10] AYED, D.; DELANOTE, D.; BERBERS, Y. Mdd approach for the development of context-aware applications. In: SPRINGER. *International and Interdisciplinary Conference on Modeling and Using Context*. [S.l.], 2007. p. 15–28.

[11] SAHAY, A. et al. Supporting the understanding and comparison of low-code development platforms. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2020. p. 171–178.

[12] RAFIQUE, W. et al. An application development framework for internet-of-things service orchestration. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 5, p. 4543–4556, 2020.

[13] GONCALVES, R. F.; MENOLLI, A.; DIONISIO, G. M. An analysis of the quality of model driven development solutions applied to cyber-physical devices. In: *Proceedings of the XXI Brazilian Symposium on Software Quality*. [S.l.: s.n.], 2022. p. 1–7.

[14] KIRUTHIKA, J.; KHADDAJ, S. Software quality issues and challenges of internet of things. *Proceedings - 14th International Symposium on Distributed Computing and Applications for Business, Engineering and Science, DCABES 2015*, IEEE, p. 176–179, 2016.

[15] ALABA, F. A. et al. Internet of Things security: A survey. *Journal of Network and Computer Applications*, v. 88, n. March, p. 10–28, 2017. ISSN 10958592.

[16] MOTTA, R. C.; OLIVEIRA, K. M. de; TRAVASSOS, G. H. On challenges in engineering iot software systems. In: *Proceedings of the XXXII Brazilian symposium on software engineering*. [S.l.: s.n.], 2018. p. 42–51.

[17] ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 10 2010. ISSN 13891286.

[18] AHMED, B. S. et al. Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study. *IEEE Access*, IEEE, v. 7, p. 13758–13780, 2019. ISSN 21693536.

[19] RECON, S. Hello Barbie Initial Security Analysis. 2016.

[20] COSTA, B. et al. Design and analysis of iot applications: A model-driven approach. In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. [S.l.: s.n.], 2016. p. 392–399.

[21] ORDOÑEZ, K.; HILERA, J.; CUEVA, S. Model - driven development of accessible software : a systematic literature review. *Universal Access in the Information Society*, Springer Berlin Heidelberg, v. 21, n. 1, p. 295–324, 2022. ISSN 1615-5297. Disponível em: <https://doi.org/10.1007/s10209-020-00751-6>.

[22] KAPTEIJNS, T. et al. A comparative case study of model driven development vs traditional development: The tortoise or the hare. *From code centric to model centric software engineering: Practices, Implications and ROI*, v. 22, 2009.

[23] AKDUR, D.; GAROUSI, V.; DEMIRÖRS, O. A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture*, Elsevier, v. 91, p. 62–82, 2018.

[24] ASGHAR, M. H.; NEGI, A.; MOHAMMADZADEH, N. Principle application and vision in internet of things (iot). *International Conference on Computing, Communication and Automation, ICCCA 2015*, p. 427–431, 2015.

[25] JAVED, A. *Building Arduino projects for the Internet of Things: experiments with real-world applications*. [S.l.]: Apress, 2016.

[26] GRIMMETT, R. *Raspberry Pi robotics projects*. [S.l.]: Packt Publishing Ltd, 2015.

[27] KONDAVEETI, H. K. et al. A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations. *Computer Science Review*, Elsevier Inc., v. 40, 2021. ISSN 15740137.

[28] PERENC, I.; JAWORSKI, T.; DUCH, P. Teaching programming using dedicated Arduino Educational Board. *Computer Applications in Engineering Education*, v. 27, n. 4, p. 943–954, 2019. ISSN 10990542.

[29] LEE, E. A. Cyber physical systems: Design challenges. In: IEEE. *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. [S.l.], 2008.

[30] SEVERANCE, C. Eben upton: Raspberry pi. *Computer*, IEEE, v. 46, n. 10, p. 14–16, 2013.

[31] NUÑEZ-VARELA, A. S. et al. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, Elsevier Inc., v. 128, p. 164–197, 2017. ISSN 01641212.

[32] XENOS, M. et al. Object-oriented metrics-a survey. In: . [S.l.: s.n.], 2000. p. 1–10.

[33] MARTIN, R. Oo design quality metrics. *An analysis of dependencies*, v. 12, p. 151–170, 1994.

[34] IMANI, M. et al. A comprehensive survey on addressing methods in the Internet of Things. *arXiv preprint arXiv:1807.02173*, 2018.

[35] RAY, B. et al. A large scale study of programming languages and code quality in github. In: ACM. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. [S.l.], 2014. p. 155–165.

[36] BURES, M. Framework for integration testing of iot solutions. *Proceedings - 2017 International Conference on Computational Science and Computational Intelligence, CSCI 2017*, IEEE, p. 1838–1839, 2018.

[37] BURES, M. et al. A Comprehensive View on Quality Characteristics of the IoT Solutions. *EAI/Springer Innovations in Communication and Computing*, n. December, p. 59–69, 2020. ISSN 25228609.

[38] KLIMA, M. et al. Selected Code-Quality Characteristics and Metrics for Internet of Things Systems. *IEEE Access*, IEEE, v. 10, n. 1, p. 46144–46161, 2022. ISSN 21693536.

[39] STAHL, T.; VOELTER, M.; CZARNECKI, K. *Model-driven software development: technology, engineering, management*. [S.l.]: John Wiley & Sons, Inc., 2006.

[40] MELLOR, S. J.; CLARK, T.; FUTAGAMI, T. Model-driven development: guest editors' introduction. *IEEE software*, IEEE Computer Society, v. 20, n. 5, p. 14–18, 2003.

[41] FAVRE, L. M. Model Driven Architecture (MDA). n. June, p. 15–33, 2010.

[42] XING, L. Reliability in internet of things: Current status and future perspectives. *IEEE Internet of Things Journal*, v. 7, n. 8, p. 6704–6721, 2020.

[43] SILVA, E. F. da; MACIEL, R. S. P.; MAGALHÃES, A. P. F. Integrating model-driven development practices into agile process: Analyzing and evaluating software evolution aspects. In: *ICEIS (2)*. [S.l.: s.n.], 2020. p. 101–110.

[44] RICCA, F. et al. On the impact of state-based model-driven development on maintainability: a family of experiments using unimod. *Empirical Software Engineering*, Springer, v. 23, p. 1743–1790, 2018.

[45] GONCALVES, R. F.; MENOLLI, A.; DIONISIO, G. M. Systematic mapping on internet of things' client-sided development. In: SBC. *Anais do XVIII Simpósio Brasileiro de Sistemas de Informação*. [S.l.], 2022.

[46] KITCHENHAM, B. et al. Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, Elsevier B.V., 2009. ISSN 09505849.

[47] PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. [S.l.: s.n.], 2008. p. 1–10.

[48] CICCOZZI, F. et al. Model-driven engineering for mission-critical iot systems. *IEEE software*, IEEE, v. 34, n. 1, p. 46–53, 2017.

[49] FLOCH, H. UML4IoT - A UML profile to exploit IoT in cyber-physical manufacturing systems. *Publication. Cayenne, French Guiana. Institut Pasteur de la Guyane francaise et de l'Inini*, v. 15, n. 348, p. 1–6, 1954. ISSN 0020-7349. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/14377658>.

[50] PANTELIMON, S.-G. et al. Towards a seamless integration of iot devices with iot platforms using a low-code approach. In: IEEE. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. [S.l.], 2019. p. 566–571.

[51] THULUVA, A. S. et al. Semantic node-red for rapid development of interoperable industrial iot applications. *Semantic Web*, IOS Press, v. 11, n. 6, p. 949–975, 2020.

[52] PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. In: . [S.l.]: Elsevier, 2015. v. 64, p. 1–18. ISSN 09505849.

[53] WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

[54] NUÑEZ-VARELA, A. S. et al. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 128, p. 164–197, 2017.

[55] DAVIS, F. D. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *Delle vicende dell'agricoltura in Italia; studio e note di C. Bertagnolli.*, v. 13, n. 3, p. 319–340, 2011.

[56] SOLINGEN, R. V.; BERGHOUT, E. W. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. [S.l.]: McGraw-Hill, 1999.

[57] HERNANDES, E. et al. Using GQM and TAM to evaluate StArt – a tool that supports Systematic Review. *CLEI Electronic Journal*, v. 15, n. 1, 2012.

[58] MCIVER, J.; CARMINES, E. G. *Unidimensional scaling*. [S.l.]: sage, 1981. v. 24.

Appendix

APPENDIX A – INTERNAL EVALUATION

A.1 SoftwareWire's Benchmark

Comments have been manually removed from the original code, while empty divider lines have been manually added to the generated code. These changes were executed in order to ease the comparison between the solutions, no other changes were made.

A.1.1 Original Master

Original source: <https://raw.githubusercontent.com/Testato/SoftwareWire/master/examples/StressTest_Master/StressTest_Master.ino>

```
#define TEST_SOFTWAREWIRE
#ifdef TEST_SOFTWAREWIRE

#include "SoftwareWire.h"

SoftwareWire myWire( A4, A5);

#else

#include <Arduino.h>
#include <Wire.h>
#define myWire Wire

#endif


void setup()
{
  Serial.begin(9600);
  Serial.println(F("\nMaster"));

  myWire.begin();
}


void loop()
```

```
{
  Serial.println(F("Test with 200 transmissions of writing 10 bytes each"));
  byte buf[20] = { 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, };
  int err = 0;
  unsigned long millis1 = millis();
  boolean firsterr = false;
  for( int i=0; i<200; i++)
  {
    myWire.beginTransmission(4);
    myWire.write( buf, 10);
    if( myWire.endTransmission() != 0)
    {
      err++;
      if( !firsterr)
      {
        Serial.print(F("first error at i = "));
        Serial.println(i);
        firsterr = true;
      }
    }
    delayMicroseconds(100);
  }
  unsigned long millis2 = millis();
  Serial.print(F("total time: "));
  Serial.print(millis2 - millis1);
  Serial.print(F(" ms, total errors: "));
  Serial.println(err);

  delay(2000);

  Serial.println(F("Sending data"));
  static byte x = 0;
  myWire.beginTransmission(4);
  myWire.write(x++);
  for( int i=0; i<random( 32); i++)
  {
    myWire.write(random(256));
  }
  int error = myWire.endTransmission();
```

```
  Serial.print(F("transmission status error="));
  Serial.println(error);

  delay(2000);

  Serial.println(F("Requesting data"));
  int n = myWire.requestFrom(4, 10);
  Serial.print(F("n="));
  Serial.print(n);
  Serial.print(F(", available="));
  Serial.println(myWire.available());


  byte buffer[40];

  myWire.readBytes( buffer, n);

  for( int k=0; k<n; k++)
  {
    if( k == 0)
      Serial.print(F("*"));
    Serial.print( (int) buffer[k]);
    Serial.print(F(", "));
  }
  Serial.println();

  delay(2000);
}
```

A.1.2 Original Slave

Original source: <https://github.com/Testato/SoftwareWire/blob/master/examples/StressTest_Slave/StressTest_Slave.ino>

```
    #include <Wire.h>

volatile byte buffer[40];
volatile int rxHowMany;
volatile int rxInterrupts = 0;
volatile boolean flagRequest;
```

```
void setup()
{
  Serial.begin(9600);
  Serial.println("\nSlave");

  Wire.begin(4);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}

void loop()
{
  noInterrupts();
  int rxInterruptsCopy = rxInterrupts;
  rxInterrupts = 0;
  interrupts();

  if( rxInterruptsCopy > 0)
  {
    Serial.print("Receive: ");
    if( rxInterruptsCopy > 1)
    {
      Serial.print("Missed:");
      Serial.print( rxInterruptsCopy);
      Serial.print(" ");
    }
    Serial.print("howMany:");
    Serial.print( rxHowMany);

    Serial.print(", data:");
    for(int i=0; i<rxHowMany; i++)
    {
      if( i == 0)
        Serial.print(F("*"));

      Serial.print((unsigned int) buffer[i], DEC);
      Serial.print(" ");
    }
```

```
    Serial.println();
  }

  noInterrupts();
  boolean flagRequestCopy = flagRequest;
  flagRequest = false;
  interrupts();

  if( flagRequestCopy)
  {
    Serial.println("Request: Data was requested and send");
  }


  noInterrupts();
  delayMicroseconds(50);
  interrupts();
}

void receiveEvent(int howMany)
{
  for( int i=0; i<howMany; i++)
    buffer[i] = Wire.read();

  rxHowMany = howMany;
  rxInterrupts++;
}

void requestEvent()
{
  static byte x = 0;


  char TXbuf[] = { 188, 0, 255, 1, 120, 150, 44, 2, 131, 72 };
  TXbuf[0] = x++;
  Wire.write(TXbuf, sizeof(TXbuf));

  flagRequest = true;
}
```

A.1.3 MDD4CPD' Master

```
#define TEST_SOFTWAREWIRE
#ifdef TEST_SOFTWAREWIRE
#include "SoftwareWire.h"

SoftwareWire myWire(32, 33);
#else
#include <Arduino.h>
#include <Wire.h>
#define myWire Wire
#endif

class Serial1
{
private:
    byte port;

public:
    Serial1(byte port)
    {
        this->port = port;
        init();
    }
    void init()
    {
        Serial.begin(port);
        myWire.begin()
    }
    void transmit(byte x, byte y, byte z)
    {
        myWire.beginTransmission(x);
        myWire.write(y, z);
        return myWire.endTransmission()
    }
    void error(i)
    {
        Serial.print(F("first error at i = "));
```

```
        Serial.println(i);
        return true
    }
};

Serial1 serial1 = Serial1(9600);

void loop()
{
    Serial.println(F("Test with 200 transmissions of writing 10 bytes each"));
    byte buf[20] = {
        100,
        101,
        102,
        103,
        104,
        105,
        106,
        107,
        108,
        109,
    };
    int err = 0;
    unsigned long millis1 = millis();
    boolean firsterr = false;
    for (int i = 0; i < 200; i++)
    {
        if (serial1.transmit(4, buf, 10) != 0)
        {
            err++;
            if (!firsterr)
            {
                firsterr = serial1.error;
            }
        }
        delayMicroseconds(100);
    }
    unsigned long millis2 = millis();
    Serial.print(F("total time: "));
```

```
Serial.print(millis2 - millis1);
Serial.print(F(" ms, total errors: "));
Serial.println(err);

delay(2000);

Serial.println(F("Sending data"));
static byte x = 0;
myWire.beginTransmission(4);
myWire.write(x++);

for (int i = 0; i < random(32); i++)
{
    myWire.write(random(256));
}
int error = myWire.endTransmission();
Serial.print(F("transmission status error="));
Serial.println(error);

delay(2000);

Serial.println(F("Requesting data"));
int n = myWire.requestFrom(4, 10);
Serial.print(F("n="));
Serial.print(n);
Serial.print(F(", available="));
Serial.println(myWire.available());

byte buffer[40];
myWire.readBytes(buffer, n);

for (int k = 0; k < n; k++)
{
    if (k == 0)
        Serial.print(F("*"));
    Serial.print((int)buffer[k]);
    Serial.print(F(", "));
}
Serial.println();
```

```
    delay(2000);
}
```

A.1.4 MDD4CPD' Slave

```
    #include <Wire.h>

volatile byte buffer[40];
volatile int rxHowMany;
volatile int rxInterrupts = 0;
volatile boolean flagRequest;

void setup()
{
    Serial.begin(9600);
    Serial.println("\nSlave");

    Wire.begin(4);
    Wire.onReceive(receiveEvent);
    Wire.onRequest(requestEvent);
}

void loop()
{
    noInterrupts();
    int rxInterruptsCopy = rxInterrupts;
    rxInterrupts = 0;
    interrupts();

    if (rxInterruptsCopy > 0)
    {
        Serial.print("Receive: ");
        if (rxInterruptsCopy > 1)
        {
            Serial.print("Missed:");
            Serial.print(rxInterruptsCopy);
            Serial.print(" ");
        }
        Serial.print("howMany:");
```

```
        Serial.print(rxHowMany);

        Serial.print(", data:");
        for (int i = 0; i < rxHowMany; i++)
        {
            if (i == 0)
                Serial.print(F("*"));

            Serial.print((unsigned int)buffer[i], DEC);
            Serial.print(" ");
        }
        Serial.println();
    }

    noInterrupts();
    boolean flagRequestCopy = flagRequest;
    flagRequest = false;
    interrupts();

    if (flagRequestCopy)
    {
        Serial.println("Request: Data was requested and send");
    }

    noInterrupts();
    delayMicroseconds(50);
    interrupts();
}

void receiveEvent(int howMany)
{
    for (int i = 0; i < howMany; i++)
        buffer[i] = Wire.read();

    rxHowMany = howMany;
    rxInterrupts++;
}

void requestEvent()
```

```
{
    static byte x = 0;

    char TXbuf[] = {188, 0, 255, 1, 120, 150, 44, 2, 131, 72};
    TXbuf[0] = x++;
    Wire.write(TXbuf, sizeof(TXbuf));

    flagRequest = true;
}
```

APPENDIX B – EXTERNAL EVALUATION

B.1 E-mail invite

Prezado {Nome}.
Sou o Rafael Figueira Goncalves, mestrando em Ciência da Computação na UEL (CCE), orientado pelo Prof.Dr. André Luís Andrade Menolli.
Considerando sua expertise em desenvolvimento de software e conhecimento sobre plataformas IoT, gostaria de pedir sua colaboração para participar de um experimento de avaliação da abordagem desenvolvida na minha pesquisa de mestrado, intitulada: MDD4CPD: abordagem de desenvolvimento dirigido a modelos para dispositivos ciber-físicos
As respostas individuais serão analisadas de forma completamente anônima. Qualquer artefato que possa vir a identificá-lo será destruído até 1 semana após a coleta.
A avaliação levará cerca de 40 minutos em uma única videochamada, onde será compartilhado um link para instruções de uso da plataforma e demais instruções para a realização do experimento.
Caso aceite participar, favor preencher este formulário.
Muito Obrigado!
Atenciosamente, Rafael,
+55 (43) 99805-1625

B.2 Questionnaire1 - Terms and Profile

Termo de aceite e Coleta de Perfil - MDD4CPD

Ao submeter este formulário, o convidado estará declarando que leu e concordou com os termos abaixo.

* Required

Termo de Consentimento Livre e Esclarecido	Você está sendo convidado(a) a participar do estudo sobre **MDD4CPD, uma Abordagem de Desenvolvimento Dirigido a Modelos para Dispositivos Ciber Físicos,** temos como objetivo **auxiliar no desenvolvimento destas soluções**, providenciando modelos multi-plataformas que facilitam o reuso, reduzindo o requerimento do conhecimento de domínio e aumentando a facilidade de sua manutenção.
Participação na Avaliação	A Avaliação tem o objetivo de testar o nível de aceitação da ferramenta em estágio de protótipo, qual já possui uma implementação mínima apresentável da abordagem MDD4CPD. A Avaliação levará cerca de 60 minutos em uma única videochamada gravada para uma análise por observação. Para realização da avaliação, um link será compartilhado para instruções de uso da plataforma e demais instruções para a realização do experimento, que inclui a réplica de uma solução dentro da ferramenta, seguido de um questionário.
Sigilo e Privacidade	Os pesquisadores garantem o respeito a sua privacidade. Qualquer artefato que possa, de qualquer forma lhe identificar, será mantido em sigilo, e destruído até 1 semana após a coleta.

1. Declaro que li e entendi todas as informações presentes neste documento, manifesto meu livre consentimento em participar no estudo, estando ciente não há nenhum valor econômico, a receber ou a pagar pela participação.

   Mark only one oval.

   - ◯ Concordo
   - ◯ Não Concordo

Perfil do Respondente

Essa seção contém questões referentes ao perfil do participante.

2. 1. Área de Formação *

Mark only one oval.

- ◯ Bacharelado na área de Computação
- ◯ Tecnólogo na área de Computação
- ◯ Licenciatura na área de Computação
- ◯ Bacharelado em outra area
- ◯ Other: _____

3. 2. Nível de formação *

Mark only one oval.

- ◯ Não Graduado
- ◯ Graduado
- ◯ Especialista
- ◯ Mestre
- ◯ Doutor
- ◯ Other: _____

4. 3. Perfil Profissional *

Check all that apply.

- ☐ Estudante
- ☐ Docente Acadêmico / Pesquisador
- ☐ Industria na área de Desenvolvimento Back-end
- ☐ Industria na área de Desenvolvimento Front-end
- ☐ Industria na área de Desenvolvimento Internet das Coisas / Ciber-Fisico
- ☐ Industria na área de Computação, outros.
- ☐ Other: _____

5. 4. Experiência com Desenvolvimento Orientado Objetos *

Mark only one oval.

- ◯ Menos de 1 ano
- ◯ De 1 ano a 3 anos
- ◯ De 3 a 5 anos
- ◯ De 5 a 10 anos
- ◯ De 10 a 15 anos
- ◯ Mais de 15 anos

6. 5. Experiência com desenvolvimento de sistemas ciber-físicos ou internet das coisas *

Mark only one oval.

- ◯ Nenhuma
- ◯ Desenvolvi projetos de complexidade baixa com Arduino ou Raspberry
- ◯ Desenvolvi projetos de complexidade média com Arduino ou Raspberry
- ◯ Desenvolvi projetos de complexidade alta com Arduino ou Raspberry
- ◯ Desenvolvi projetos de complexidade alta em diferentes plataformas Iot, não apenas Arduino ou Raspberry

7. 6. Se respondeu diferente de "nenhuma" na questão 5, qual a sua experiência com sistemas ciber-físicos ou internet das coisas?

Check all that apply.

- ☐ Projetos Pessoais
- ☐ Projetos Acadêmicos
- ☐ Projetos em Nível Profissional
- ☐ Other: _____

8. 7. Algum comentário relevante que queria fazer sobre o seu perfil.

Percepção de Importância

Essa seção contém questões referentes a suas percepções sobre alguns aspectos relacionados ao desenvolvimento de software. Assinale de 1 a 5 informando o nível de concordância com cada afirmação.

9. 8. O uso de artefatos de **Modelos Diagramáticos** no desenvolvimento de software é importante. *

Como exemplos de Modelos Diagramáticos, tem-se diagramas de classes, casos de uso, atividades entre outros.

Mark only one oval.

	Não concordo					Concordo
	1	2	3	4	5	

10. 9. As **restrições de regras de negócio** providenciadas por um modelo, são artefatos importantes no desenvolvimento de software. *

Por exemplo ao enforçar o princípio de instância unica ao utilizar o padrão singleton.

Mark only one oval.

	Não concordo
1	◯
2	◯
3	◯
4	◯
5	◯

Concordo

11. 10. A **Programação Orientada a Objetos** é um paradigma importante no desenvolvimento de software. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

12. 11. A **organização** do código--fonte é um fator importante no desenvolvimento de software. *

Mark only one oval.

	Não concordo					Concordo
	1	2	3	4	5	

13. 12. Um ambiente de desenvolvimento possuir **detecção de erros** é um fator importante no desenvolvimento de software. *

Detecção de erros em tempo de edição da solução, antes dos tempos de compilação e execução.

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

14. 13. O ambiente de desenvolvimento possuir: **documentação de fácil acesso, descrevendo os métodos e atributos dos componentes utilizados**, é um fator importante no desenvolvimento de software. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

15. 14. O tempo de **geração de código** é um fator importante no desenvolvimento de software. *

Um exemplo de rápida geração é automático e instantâneo assim que o modelo sofrer alterações, em contraste com um exemplo de geração demorada, onde o processo seria requisitado manualmente e gerado após alguns minutos.

Mark only one oval.

	Não concordo
1	◯
2	◯
3	◯
4	◯
5	◯

Concordo

B.3 Questionnaire2 - Technology Acceptance Model

MDD4CPD

Este é o questionário para a avaliação da ferramenta.

* Required

Facilidade de uso da ferramenta

Essa seção contém somente questões referentes ao uso da ferramenta.
Assinale de 1 a 5 informando o nível de concordância com cada afirmação.

1. 1. Em geral, foi fácil aprender a usar a ferramenta. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

2. 2. Tive facilidade na manipulação do diagrama. *

Diagrama se trata de todo o canvas entre a paleta de componentes e o bloco de geração de codigo.

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

3. 3. Eu entendi como foi feito o processo de criação dos componentes nós. *

Componentes nós são aqueles representados na paleta de componentes, uma vez no diagrama podem ser conectados entre si com links.

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

4. 4. Eu entendi a interação entre os componentes nós, realizado por links. *

Mark only one oval.

	Não concordo					Concordo
	1	2	3	4	5	

5. 5. Os erros e problemas apontados pela ferramenta auxiliaram o processo de desenvolvimento. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

6. 6. Eu obtive sucesso na geração da solução "blink" de exemplo. *

Mark only one oval.

	Não concordo				Concordo
	1	2	3	4	5

Utilidade

Essa seção contém somente questões referentes ao uso da ferramenta.
Assinale de 1 a 5 informando o nível de concordância com cada afirmação.

7. 7. O uso da ferramenta auxilia a gerar a solução. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

8. 8. O uso da ferramenta reduz a necessidade do conhecimento de domínio. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

9. 9. O uso da ferramenta auxilia na detecção e prevenção de erros. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

10. 10. O uso da ferramenta auxilia na organização do código. *

Mark only one oval.

Não concordo

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Concordo

Documentação

Essa seção contém somente questões referentes a documentação utilizada neste experimento.
Assinale de 1 a 5 informando o nível de concordância com cada afirmação.

11. 11. A documentação foi um item importante para o uso da ferramenta. *

Mark only one oval.

	Não concordo
1	◯
2	◯
3	◯
4	◯
5	◯
	Concordo

Seção para comentário adicional

12. 12. Algum comentário que queria fazer sobre a ferramenta.

B.4 MDD4CPD User Manual

Most of the images on the user manual are animated, which aren't supported on PDF, an layer of transparency has been added to the images for printing.

Manual do Usuário MDD4CPD

Introdução

MDD4CPD é uma abordagem de desenvolvimento dirigida a modelos, para dispositivos ciber-físicos.
A abordagem foi criada visando auxiliar no desenvolvimento de software neste domínio, por meio de:

- Facilidade de reutilização de modelos em múltiplas plataformas.
- Facilidade na manutenção das soluções.
- Modelagem final é realizada em um único passo e por meio de um diagrama.
- Lista de componentes facilmente extensível pelo desenvolvedor, podendo importar bibliotecas.
- Detecção de erros (erros que seriam detectados somente em tempo de compilação).
- Transformação em código-fonte é feita instantaneamente após cada mudança do diagrama.
- A transformação em código-fonte é completa, não precisando de alterações.

(estes pontos são em especial contraste com abordagens atualmente disponíveis)

Dispositivos ciber-físicos são os componentes que integram sensores, computação, controle e comunicação em objetos físicos, podemos considerar os elementos no da Internet das Coisas.
O termo ferramenta neste manual se refere ao software de protótipo, produzido com o intuito de validar a abordagem.
Aqui está um vídeo breve apresentando os diferenciais da ferramenta:
https://youtu.be/iEBqGXhNCIc

Limitações Atuais da Ferramenta

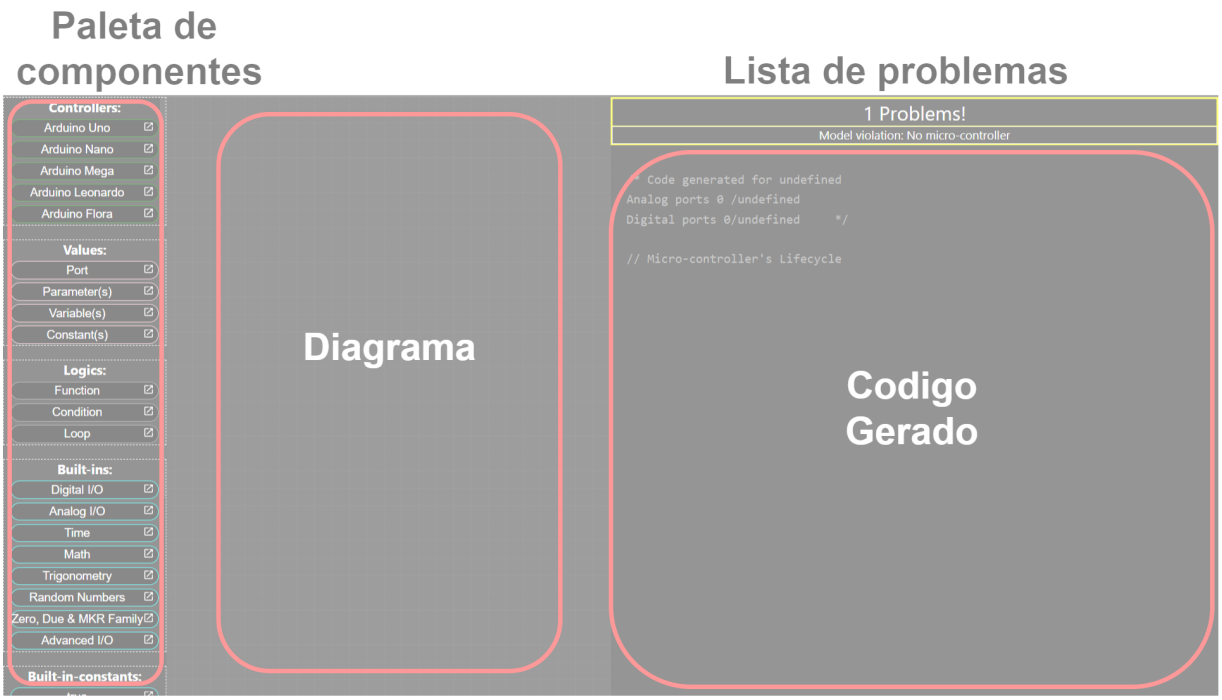
A ferramenta contém uma especificação mínima da abordagem, tendo ainda uma série de problemas atuais, que não estão sendo resolvidos neste primeiro momento por uma questão de tempo.

Limitação	Futura solução
Alta complexidade visual observada em exemplos simples como o "blink".	Adicionar níveis de abstração opcionais, onde o zoom do diagrama afetará um agrupamento de componentes, como em sub-processos do BPMN.
Confusão visual ao ter mais de uma conexão na mesma porta, ambiguidade ao saber qual a ordem das conexões	Manter uma coordenação das cores utilizadas nos links. Adicionar números aos links para indicar/editar a ordem das conexões

Interface Gráfica

A interface gráfica da ferramenta se divide em 4 segmentos:

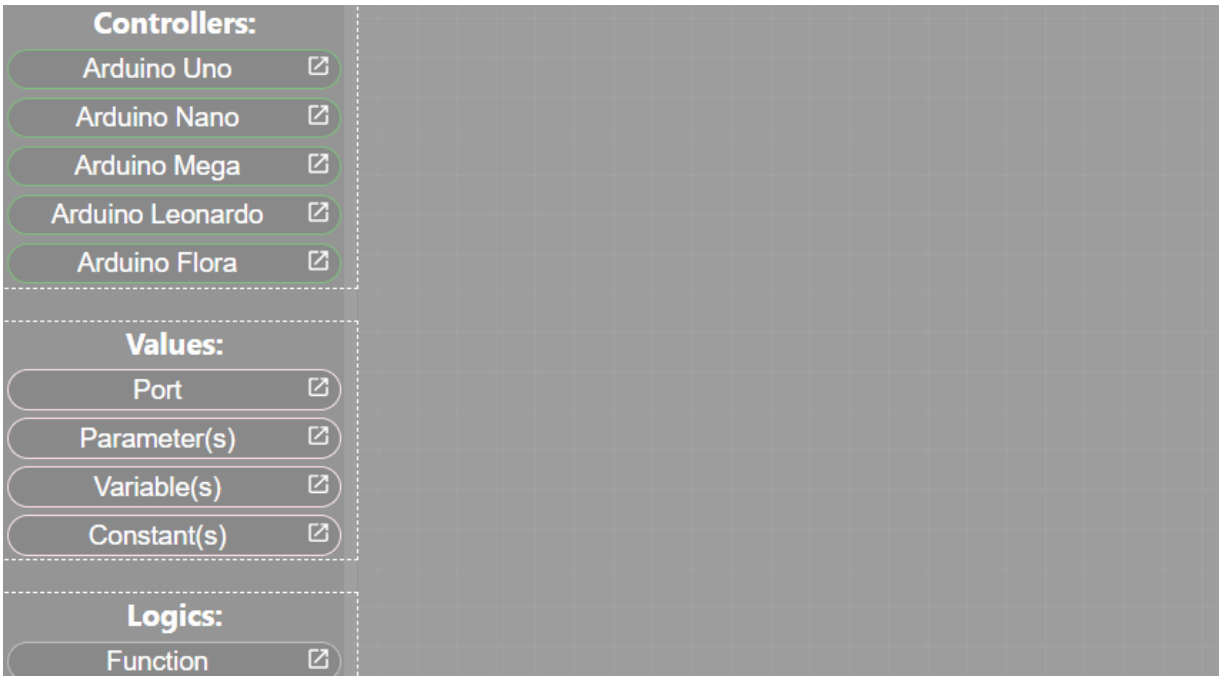
1. Paleta de componentes
2. Diagrama
3. Código Gerado
4. Lista de problemas



Paleta de componentes

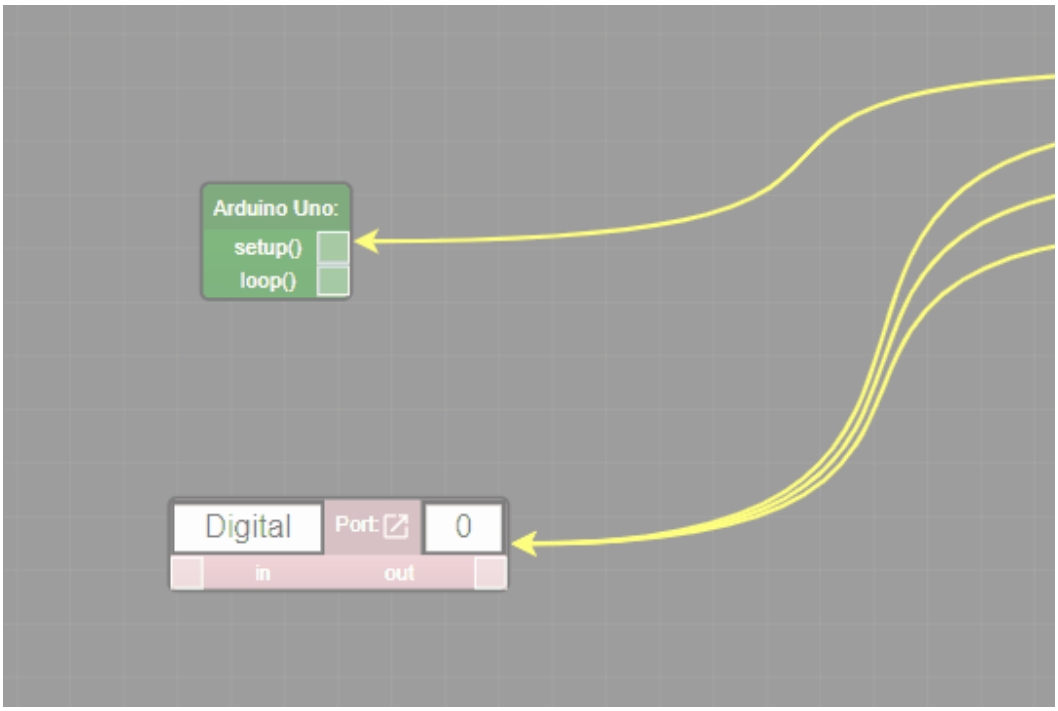
Possui todos os componentes utilizados no diagrama, cada componente possui um ícone que ao passar o mouse vai mostrar uma janela com informações extra de cada componente, como o seu meta-modelo e o texto explicativo.
Cada componente na lista foi gerado a partir de um meta-modelo..

Arrastando componentes da paleta para diagrama resulta no aparecimento do node relevante:

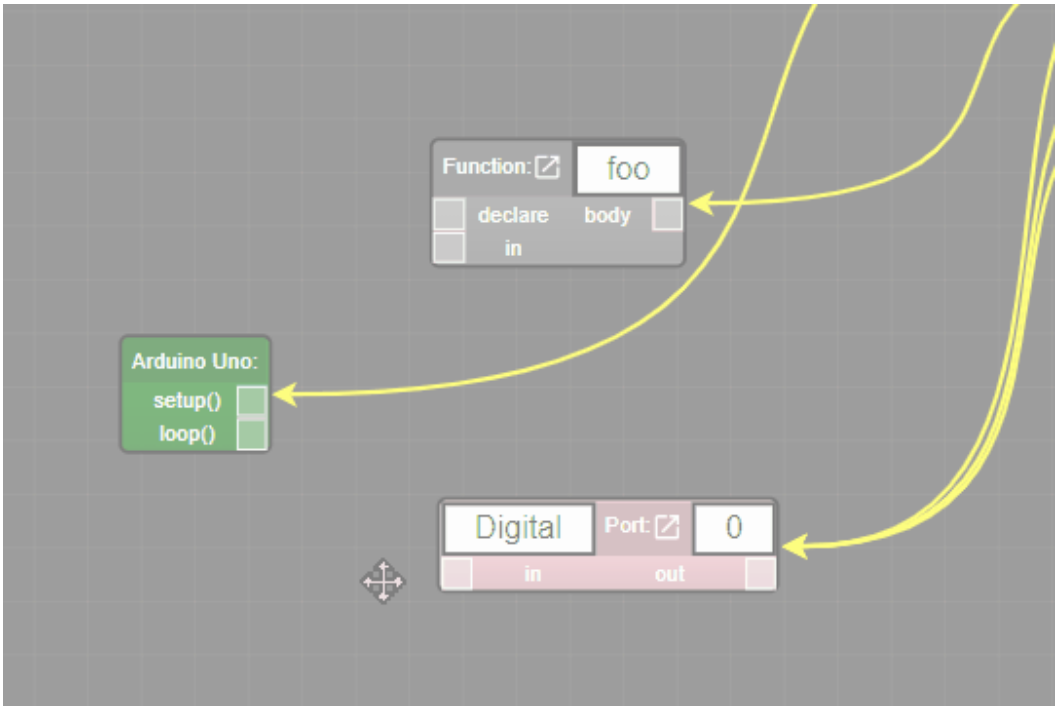


Diagrama

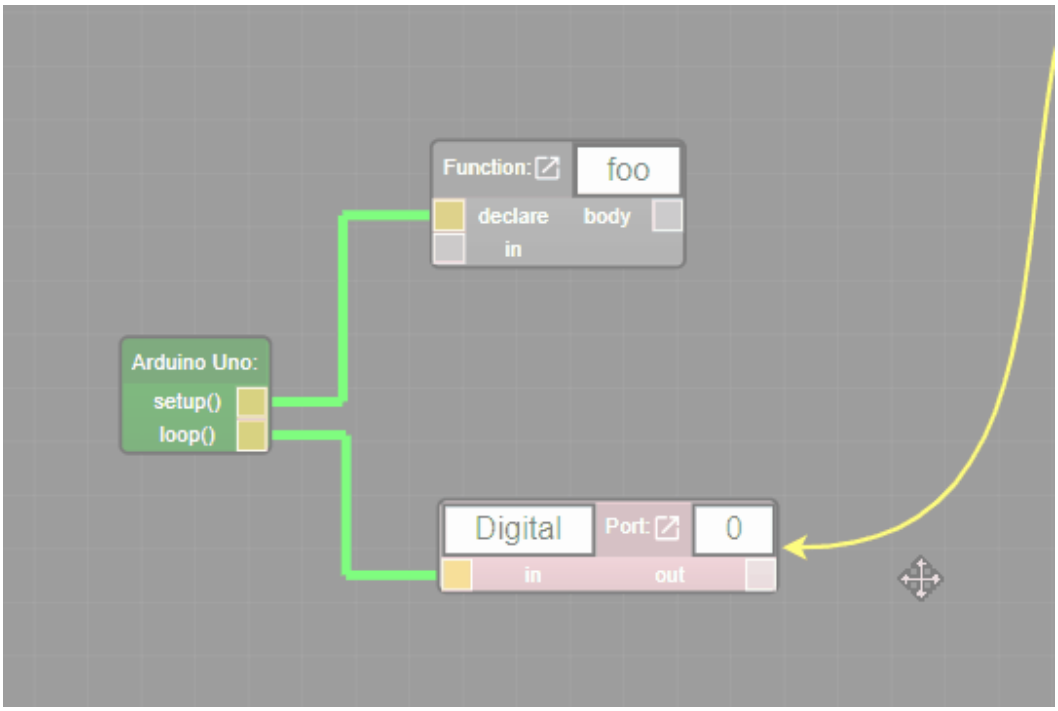
O diagrama possui somente dois tipos de itens, os componentes em formato de node e as conexões entre os mesmos.
Arrastando nodes para melhor organização visual:



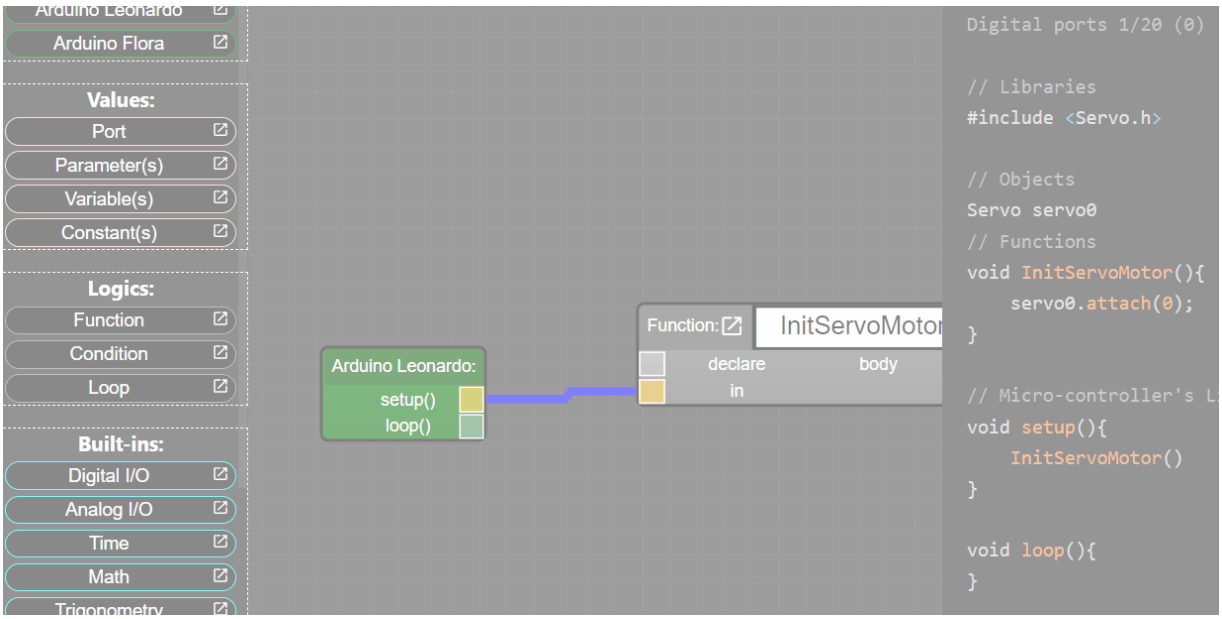
Links são criados ao arrastar a partir de uma porta de entrada ou saída, sua funcionalidade vai depender dos nodes específicos:



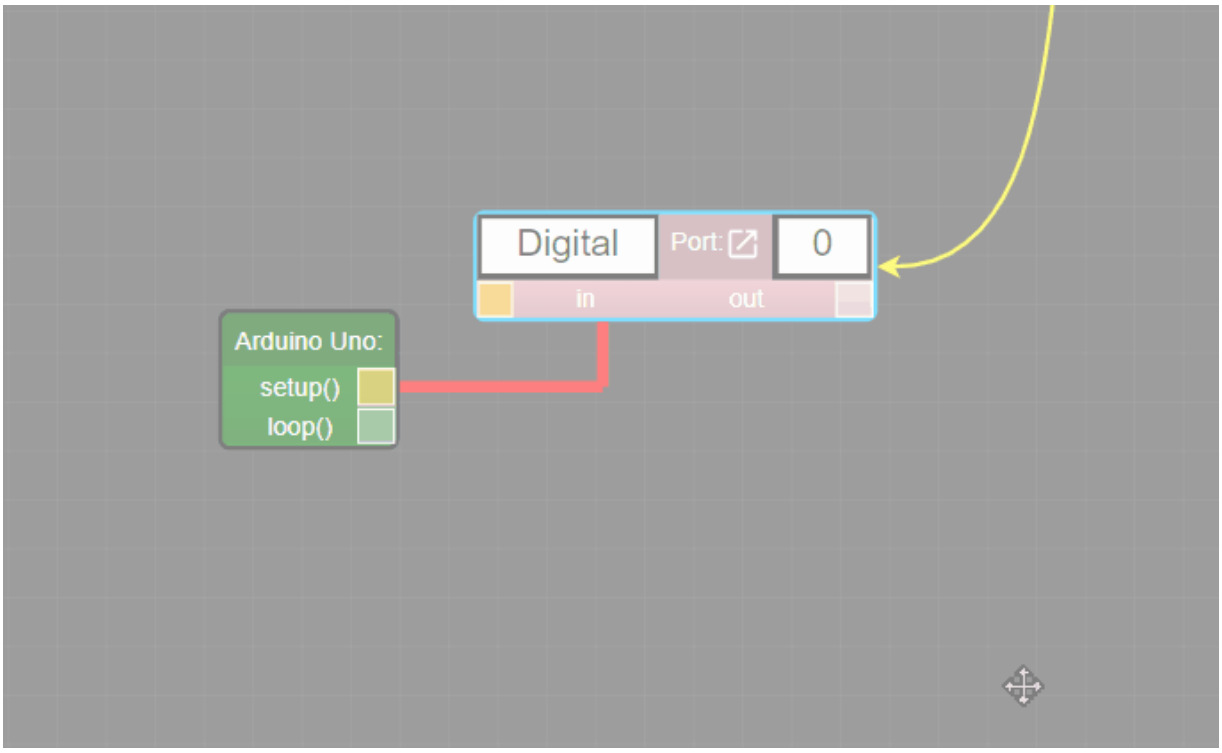
Links e nodes devem ser selecionados com o cursor antes de serem deletados, a partir disso basta teclar “Delete” no teclado.



O diagrama pode ser navegado por meio do arrasto, zoom é controlado por meio do scroll:

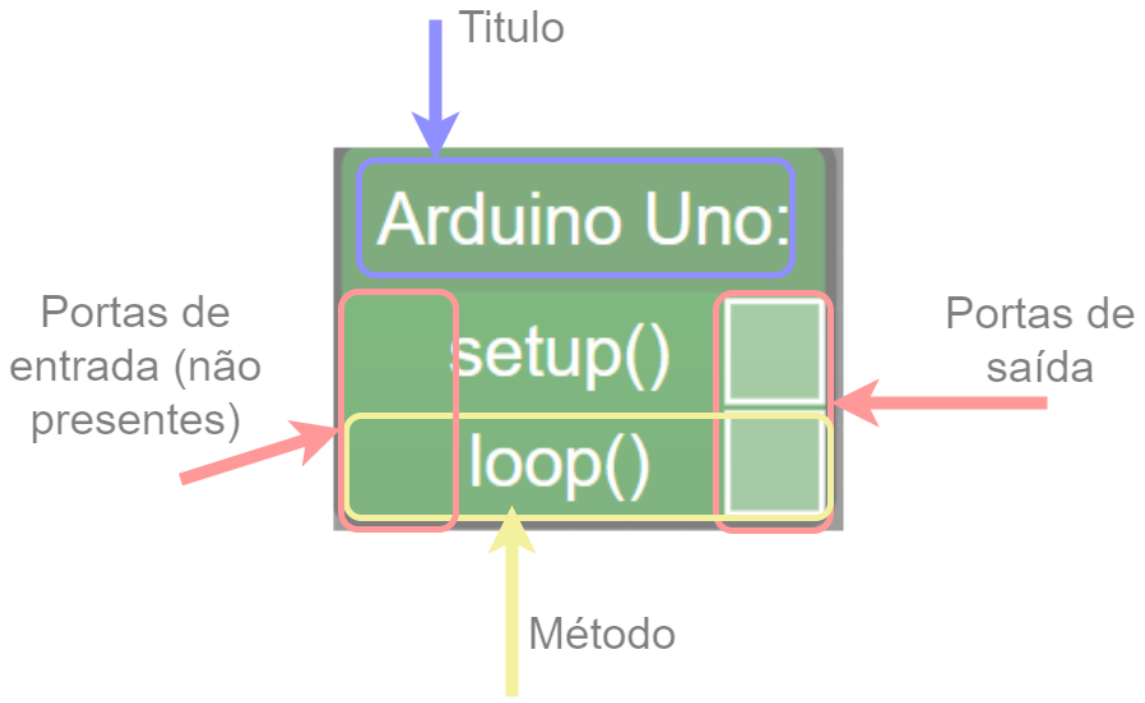


Links podem ter sua geometria alterada para melhor organização visual, para isso o link deve estar selecionado, a partir desse momento qualquer segmento dele pode ser arrastado:



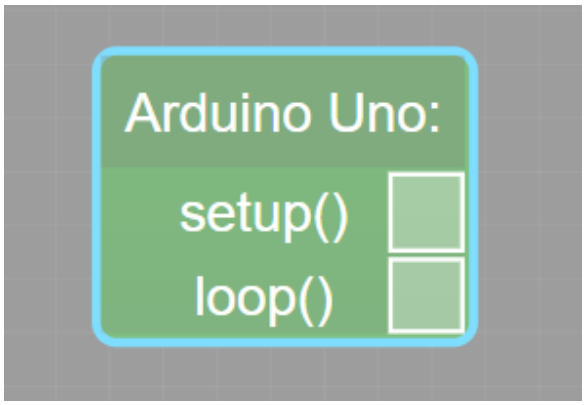
Node

Este é a representação de um node:

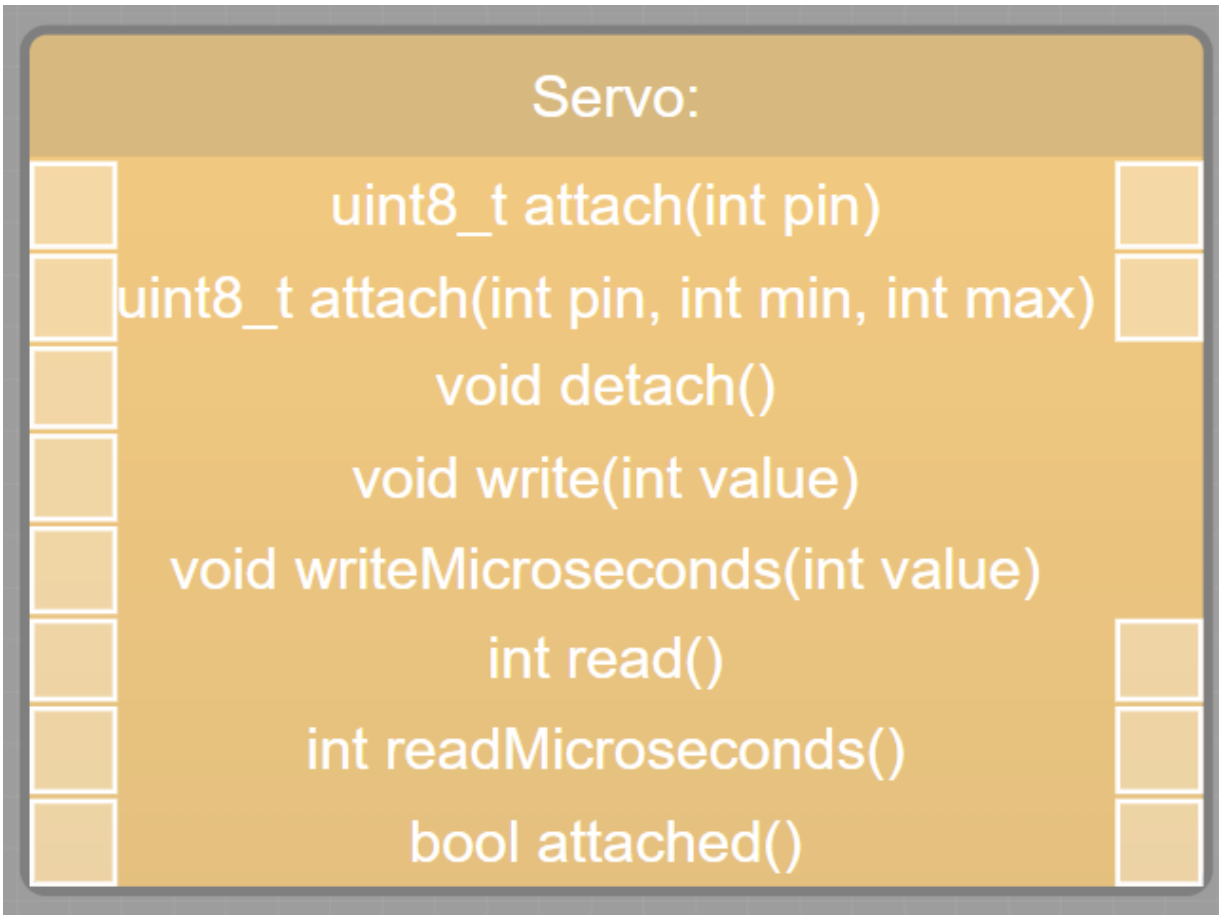


No caso, o Arduino Uno possui somente 2 métodos, ambos não possuem retorno, tendo somente portas de saída.

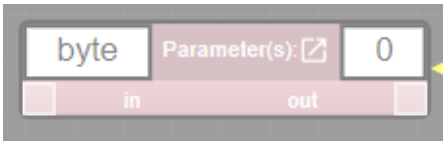
Este é um node selecionado, ao apertar “Delete” esse node será deletado:



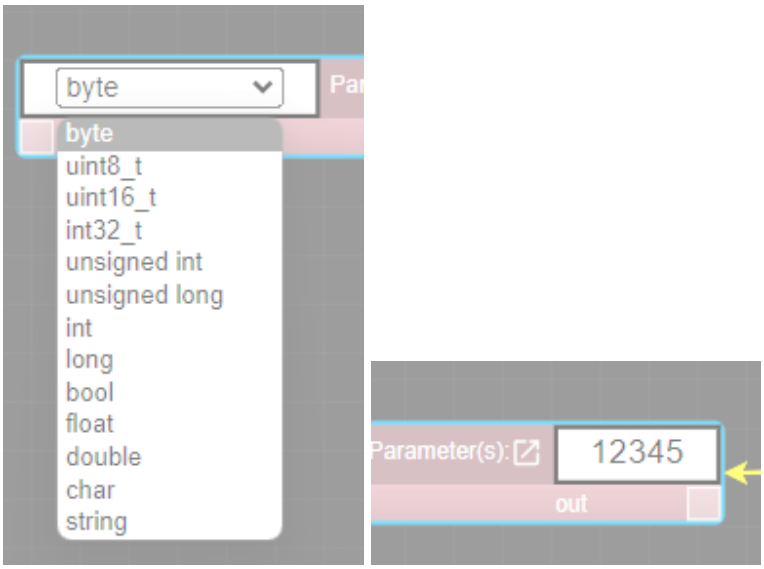
Aqui está um exemplo de no com métodos que possuem entrada e saída.
Nota-se que somente métodos do tipo void, que não possuem retorno, não possuem saída.



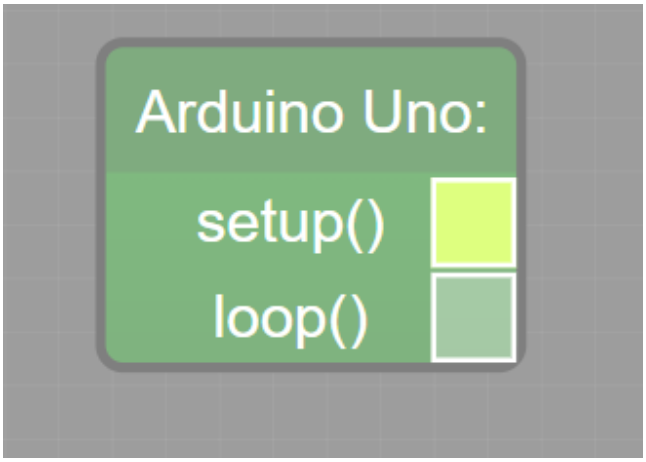
Existem nodes com campos de informações adicionais como valor, nome e tipo:



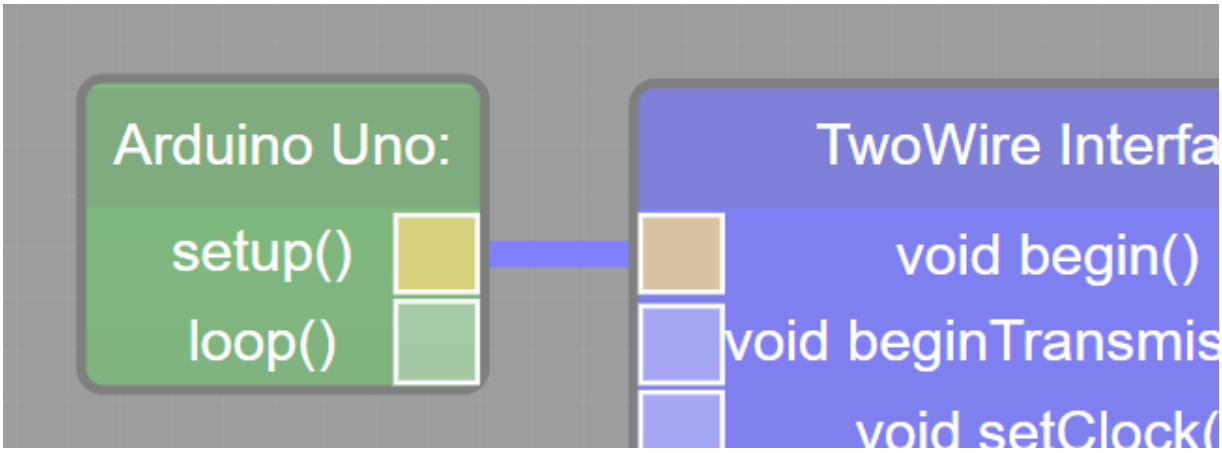
Basta clicar duas vezes rapidamente em cima de um desses campos, ele entrará em modo de edição, onde poderá ser selecionado ou digitado um novo valor, para aplicar a alteração basta teclar “Enter”.



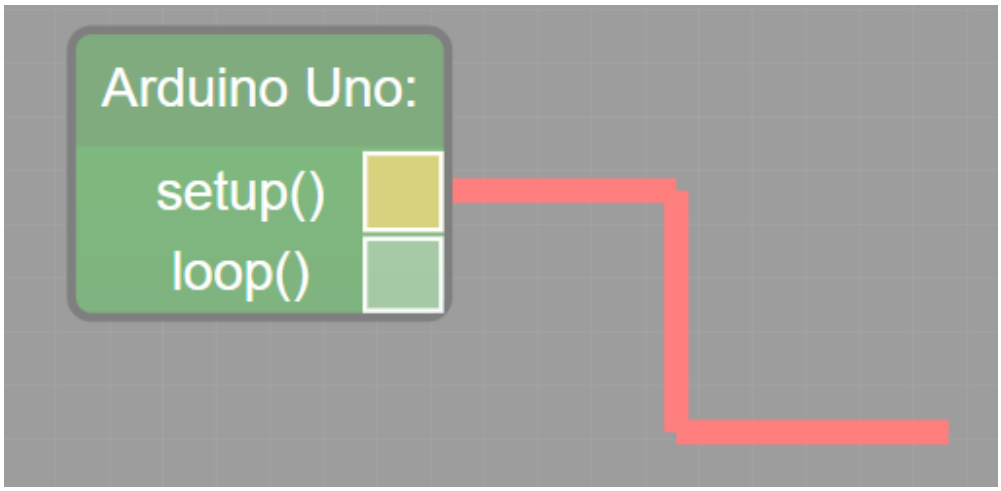
Links



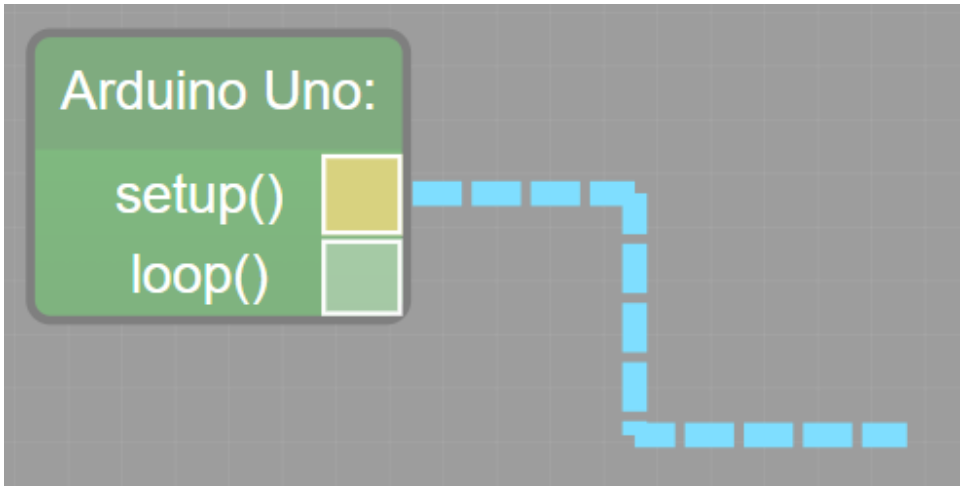
Ao passar com o mouse por cima de uma porta de entrada ou saída, a mostra uma alteração de cor



Ao clicar e arrastar até outra porta, será criado um link entre as duas portas



Caso um node não seja conectado, ou indesejado, ele precisa ser deletado, pode ser selecionado ao ser clicado, ao clicar delete ou backspace esse link será deletado



Código Gerado

O código apresentado na ferramenta é gerado conforme a modelagem do diagrama a sua esquerda, existe com o intuito de ser copiado e colado na IDE/Simulador de escolha.
Atualmente não existe reinterpretação de do código gerado, assim qualquer alteração realizada após a extração não será sincronizado com o diagrama.
O código gerado pode ser inválido caso existam problemas apontados pela ferramenta.

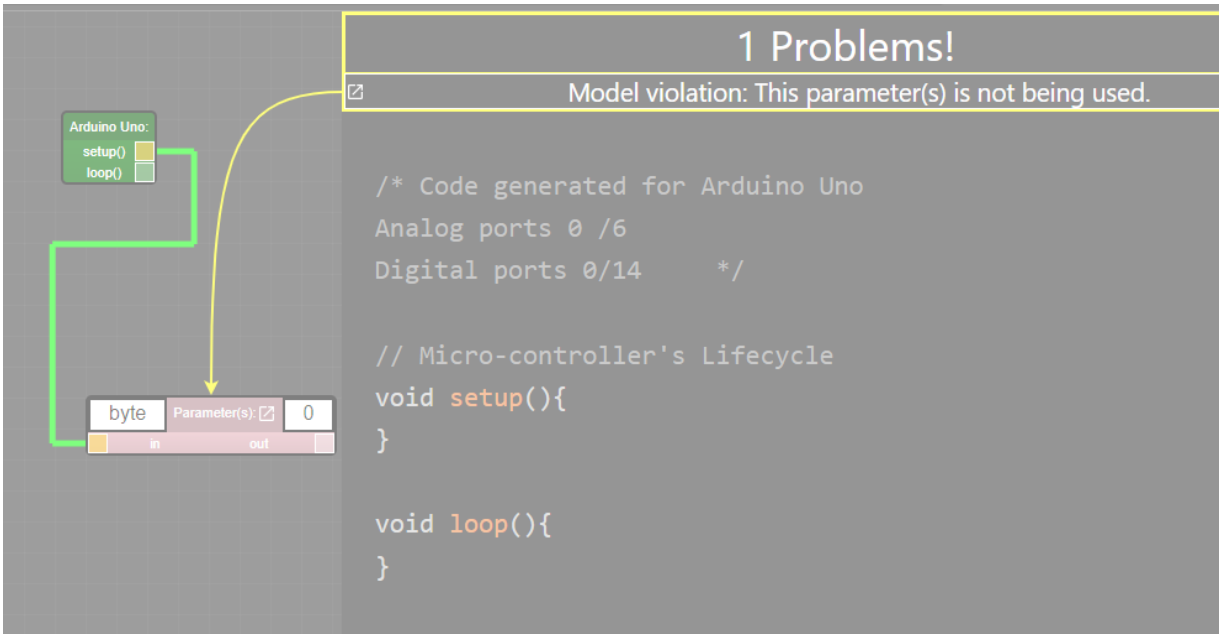
```
/* Code generated for Arduino Uno
Analog ports 0 /6
Digital ports 0/14     */

// Micro-controller's Lifecycle
void setup(){
    pinMode(LED_BUILTIN,OUTPUT);
}

void loop(){
    digitalWrite(LED_BUILTIN,HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN,LOW);
    delay(1000);
}
```

Lista de problemas

Este segmento existe para auxiliar a modelagem da sua solução, apontando violações do modelo.
Violações podem ser erros que impediam a execução do código gerado, assim como avisos de variáveis declaradas não sendo utilizadas, como linters modernos.



Exemplo de Implementação

O projeto “Blink” é considerado como o Hello World da programação do Arduino, utilizado para demonstrar uma solução extremamente simples a ponto de testar o ambiente de desenvolvimento e hardware.
O script faz com que um LED ligue e desligue de maneira constante (muitos arduinos possuem um LED built-in, embutidos na própria placa, não precisando nem de um LED externo para a execução. Video

APPENDIX C – SYSTEMATIC MAPPING

Systematic Mapping on Internet of Things' Client-Sided Development

Rafael Figueira Goncalves
rafael.goncalves@uel.br
Universidade Estadual de Londrina
UEL
Brazil

André Menolli
menolli@uenp.edu.br
Universidade Estadual do Norte do Paraná UENP
Universidade Estadual dde Londrina
UEL
Brazil

Gustavo Marcelino Dionisio
gustavo.dionisio@uel.br
Universidade Estadual de Londrina
UEL
Brazil

ABSTRACT

Context: the Internet of Things (IoT) is a paradigm that provides an ecosystem for a fast-growing quantity of connected devices, also defined as cyber-physical devices. Problem: the creation of Internet of Things solutions is fairly complex, having to integrate and communicate between sensors, devices, and larger systems, presenting many technical challenges not present in the same magnitude as other paradigms. One of the most affected segments is the development of cyber-physical devices. Much of its development energy is spent on the connecting and efficacy of these devices, often overlooking the future impacts of the proposed solution, caused by a lack of software quality. Solution: The execution of a Systematic Mapping in order to bring attention to possible research gaps. SI Theory: This work follows the accepted protocols for systematic mappings, meta-analysis, and Hermeneutics. Methodology: this paper executes a systematic mapping, following well-accepted guidelines in order to systematically gather, include and classify scientific papers according to IoT devices' own characteristics. Results: 8146 studies were found and reduced to 211 relevant studies that focused on client-side IoT development had their data graphed and analyzed. Our results show a lack of software metrics used, many research gaps and correlations were discovered, when in respect to specific software quality properties as described by the ISO25010 and other characteristics collected, such as programming languages and study domain. Contributions: The main contribution of this study is to expose multiple research gaps present in IoT client-side development. Providing a background for future information system studies on techniques and tools to improve IoT development.

CCS CONCEPTS

• **General and reference → Metrics**; • **Computer systems organization → Embedded and cyber-physical systems**; • **Hardware → Emerging tools and methodologies**.

SBSI 2022, May, 2022, Curitiba, Brazil

ACM ISBN 00…$00
https://doi.org/10.1145/1234567890

KEYWORDS

Internet of Things, Software Quality, Software Metrics

ACM Reference Format:
Rafael Figueira Goncalves, André Menolli, and Gustavo Marcelino Dionisio. 2022. Systematic Mapping on Internet of Things' Client-Sided Development. In *Proceedings of the XVIII Brazilian Symposium on Information Systems (SBSI 2022)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1234567890

1 INTRODUCTION

The quantity of research and commercial applications involving internet-connected devices has grown exponentially, with an increase of 1125%, from 2000 to 2019 to the percentage of global internet access int [1]. Following this trend is the Internet of Things (IoT), as stated by Howell [11] the number of connected devices (excluding computers and smartphones) will go from 27 billion in 2017 to a projected 125 billion before 2030. IoT is proving itself to be a prominent technology in the near future.

Asghar et al. [3] define IoT as an extension of the internet in the real world, by connected objects. Atzori et al. [4] produced a survey exposing the technology's main challenges, their simplest definition being the concept of universal presence, an object having to fulfill its purpose in an abstruse manner.

IoT is capable of a variety of different applications, Gubbi et al. [10] argues that these solutions are ideal for promoting innovations, being limited only by the creativity of its makers. These solutions are able to embed computational intelligence into different devices, allowing for more practical and useful environments, such as smart homes and offices having the ability to collect and deliver data to their users.

IoT has a very low barrier to entry, partially due to highly popular, mostly open-source prototyping platforms, such as Arduino ([14] and Raspberry Pi [9]. These are used in multiple initiatives for educational purposes but are also used in commercial applications.

These single board solutions facilitate prototyping before mass production, where the whole solution is usually reduced to its indispensable components in a custom board. The high popularity of these platforms is also attributed to the vast component selection, ease of use, and community support as stated by Kondaveeti et al. [17].

The components and the accompanying libraries are used for more than just prototyping, usually making their way to commercial devices. These libraries and companion documentation are usually a mixture between provided by the manufacturer and community effort, and are freely available to anyone in a code-sharing

platform such as GitHub. Customarily these lack standardization (also noticed by Coetzee and Eksteen [6]), the use of good practices, and even a low harnessing of the technology, such as not utilizing Object-Oriented Programming in the C++ Language. The low software quality present in these solutions is representative of an area of interest in this systematic mapping.

Regardless of widespread availability and easy-to-use platforms, IoT solutions, however, are intrinsically not simple. They may potentially be more complex when compared to better-established paradigms such as desktop or mobile. IoT involves different physical devices, such as servers and network switches which are already present in other paradigms. However, the development of these heterogeneous "things" also known as cyber-physical devices is where most of these novelty challenges are found.

Some of the challenges specially present in client-sided devices are for example: the low computational power in these node devices; the high vulnerability for security issues mentioned by Alaba et al. [2]; the higher requirements for speed; and a previously unmatched concern with energy consumption while requiring high availability.

Considering the introduced material, this work presents a systematic mapping with the objective of investigating findings on how (client-sided) IoT development is executed, with attention given regarding software quality.

It is apparent the existence of secondary works addressing IoT in the context of quality of service and product quality, however, there were no works that seek to identify the state of the art of software quality for the IoT. This systematic mapping has the objective to answer the following research question: What are the main aspects involved in the development of IoT solutions? Moreover, we are especially interested in the software quality of these solutions. Which methodologies are being applied in order to maintain quality, and which quality characteristics have more attention devoted to than others?

2 BACKGROUND

This section presents an overview of software quality, an essential topic to this systematic mapping. Related work is discussed at the end of the section.

2.1 Software Quality

The application of software metrics is a way to quantify quality by the selected criteria Nuñez-Varela et al. [23]. Xenos et al. [31] determined that the selection of which software metrics should be used is strongly related to the concerned project since the product from an unsuitable metric will be of no value to the analysis.

In order to analyze the studies, a more systematic view on software quality was needed, out of the current quality models available the ISO 25010:2011 [13] was chosen. The ISO 25010:2011 is a revised model on software quality requirements and evaluations, built on an international consensus. The model defines a set of quality attributes, these are categorized into eight characteristics shown in Fig 1. All the problematic areas listed by Imani et al. [12] are embraced by the ISO, reassuring the model's relevance for IoT development.

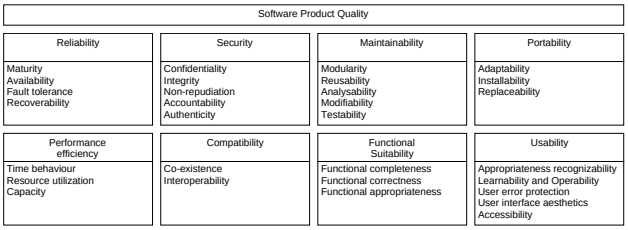


Figure 1: ISO 25010's eight characteristics [13].

2.2 Related works

There are secondary works in the literature that investigate IoT and its approaches and techniques. This section aims to identify recent works related to this systematic mapping.

Machado et al. [21] performed a systematic mapping with the objective to identify characteristics, sub-characteristics, and product quality measures for smart-city applications. Thus, the article sought to show the state of the art on product quality for smart cities. It was observed that there is no consolidated model to assess product quality for smart city applications. In addition, of the 47 quality properties found, 33 are in accordance with ISO / IEC 25010: 2011, and the other 14 have been identified as new properties.

White et al. [29] produced a systematic mapping on service quality approaches in IoT, in order to identify the approaches used and possible research gaps. It was identified that the communication layers in the IoT architecture are the most addressed. Regarding quality, it was noted that reliability, performance, and stability are the most addressed factors. On the other hand, it was identified that some important factors for the success of IoT are seldom used, such as security, compatibility, and maintainability.

The systematic mapping carried out by Cortés et al. [7] focuses on the adoption of software testing for the IoT. It was found that the most common tests assess for: performance, usability, functionality, system, user, and stress. It was identified the presence of a difficulty to perform system tests in IoT due to the factors of heterogeneity, distribution, and scalability, pointing to an area to be further explored by the literature.

Other systematic mapping works on IoT were found with an indirect focus on quality. Nguyen et al. [22] focused their systematic mapping on orchestration and deployment approaches for IoT. This area is a challenge since devices of a dynamic and heterogeneous nature are involved. The mapping presents how little can be found about this area of the literature, and how in recent years, scientific studies have started addressing it. It was also identified that there is a lack of work that addresses orchestration and implantation in low-level IoT devices. Wolny et al. [30] produced an initial systematic mapping with the following goal: to verify what is the state of the art of Model-Driven Engineering for IoT (MDE4IoT). With this, several techniques were mapped by the study, the most recurring in decreasing order: Semantic / Meta Model, Specific Framework, Code Generation, and Domain-Specific Languages.

3 METHODOLOGY

This section formalizes the guidelines followed, the research questions, study selection, search string, and the quality assessment of

this systematic mapping. In order to achieve this work's objective the up-to-date guidelines for systematic mapping studies in software engineering from Petersen et al. [26] were followed, which contrasts their previous work from 2008 to the current researches, validating their findings against the acclaimed Kitchenham et al. [16] and Kitchenham et al. [15], which is also cited directly in this mapping.

3.0.1 Research Questions. The purpose of this mapping study is to investigate the extent to which software quality is prioritized by IoT developers, what methodologies are applied, and how software quality influences IoT development in the client-side implementations. Following the guidelines from Kitchenham et al. and Petersen et al. The following research questions were elaborated:

- RQ01: How often is every quality requirement cited in the analyzed articles.
  By quality requirements we use ISO25010 ISO 25010:2011 as a guide, looking for the following categories of quality: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability.
- RQ02: Which software metrics, if any, are used for quality validation.
  Any software metric is being gathered, no discrimination is being applied, for example, software metrics only applicable to certain programming paradigms.
- RQ03: What type of experiment is being used for quality validation.
  By validation the expected experiment is either internal or external testing of the proposed approach.
- RQ04: Which platform, domain, framework and/or paradigm is being used.
  Platform, domain, framework, and paradigm are all embrace the broader definition of each term, expecting, for example, hardware and software development platforms, programming and network platforms and so forth, such interpretation carries the necessity of further sorting of the results into auxiliary collections.

3.0.2 Study selection. The following digital databases were used to search and filter studies: Science Direct, IEEE, ACM, and SpringerLink. Fig 2 shows acceptance numbers by database in addition to an overview of the mapping process. We excluded articles based on titles and abstracts, as well as full text reading whenever inconclusive. The inclusion criteria were evaluated by three different researchers, with a third researcher serving as a tiebreaker in the case of any disagreements. This was done to ensure the reliability of the mapping study.

The following inclusion criteria were applied:

- Studies in the English language.
- Primary studies
- Studies in the field of Software Engineering.

The following exclusion criteria were applied:

- Books and grey literature.
- Studies which are duplicates of other studies.
- Does not deal with "things"-oriented development.

In Fig 3 it is displayed the reasons for the 7819 rejections.

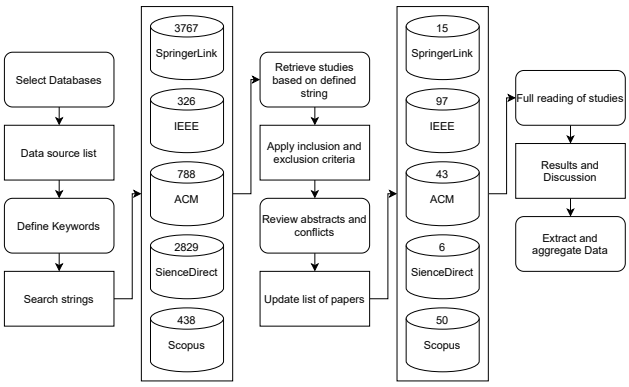


Figure 2: Research method representation inspired by Geraldi et al. [8].

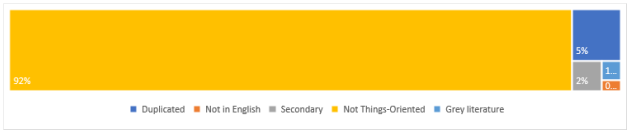


Figure 3: Reason for rejections.

3.0.3 Search String. The search string was constructed with the objective of fetching all IoT content related to quality measurement:

> ("iot" OR "internet of things" OR "smart devices" OR "smart objects" OR "internet of everything" OR "machine to machine" OR "ambient intelligence" OR "smart dust" OR "cyber-physical" OR "smart city" OR "smart cities") AND ("programming quality" OR "programming metric" OR "programming analysis" OR "programming evaluation" OR "programming measurement" OR "software quality" OR "software metric" OR "software analysis" OR "software evaluation" OR "software measurement" OR "code quality" OR "code metric" OR "code analysis" OR "code evaluation" OR "code measurement" OR "code performance" OR "software performance" OR "performance testing")

3.0.4 Quality assessment. The number of included and excluded articles is shown in Fig 4 at each stage. Cohen's Kappa (κ) coefficient Landis and Koch [19] was used to measure the inter-rater agreement between the two first researchers in their evaluation. The Kappa coefficient was 0.614, indicating a substantial agreement between the researchers (95.88%). According to Landis and Koch [19] these numbers indicate a low likelihood of random agreement.

During the full reading of the studies, some articles were eliminated based on the established criteria for inclusion and exclusion. The final count of studies was 253 and their quality was evaluated by responding to a set of questions:

- Have the authors cited applying in their research an existing or new IoT platform? If so which.
- Have the authors cited applying in their research an existing or new IoT Framework?

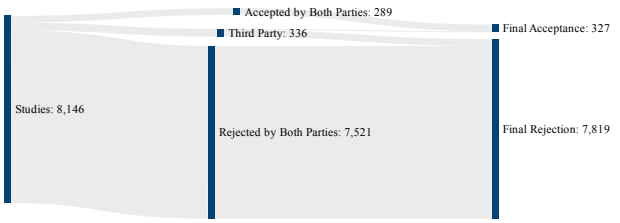


Figure 4: Studies acceptance process.

- Have the authors cited programming paradigm? if so which?
- Have the authors cited the programming language(s) used? If so which?
- Have the authors made their work publicly available whenever applicable? All source code available has been gathered for future analysis.
- Have the authors cited being worried or taking into account software quality? If so which qualities were cited?
- Have the authors cited conducting any type of experiment for validating their approach? If so which?
- Have the authors cited using software metrics as a method of validating software quality? If so which?

The quality assessment was conducted by an individual author fully reading all studies that passed the inclusion and exclusion criteria, resulting in 211 accepted studies.

4 SNOWBALLING PROCESS

On the account of this systematic mapping having had a couple of different iterations. An initial version was submitted to another conference and included the snowballing process, however it was suggested to improve the search string. Thus, for this submission the work has been improved throughout, one of these improvements was the addition of different terms suggested by the reviewers, to the search string. Alongside with the necessity of re-running the search in all databases in order to update the time interval, the opportunity was taken be even more extensive in the string building process, adding more terms, including the ones needed to arrive at our previous snowballing results, which is now guaranteed to be a subset of the collected articles.

5 MAPPING RESULTS

This section presents the execution of the analysis and discussion on the results of the systematic mapping. The goal is to answer the four research questions presented in Section 3, extracting any additional information by exploring the results, demographics, and trends from the mapped studies. The list of accepted studies is available here due to the size constraints.

5.1 Characterization of studies

This subsection presents an overview of the 211 selected studies' general information, including period and venue distribution.

5.1.1 Distribution of studies. Regarding the type of venues, Conferences were by far the most popular, as presented in Fig 5 the distribution of studies per venue.

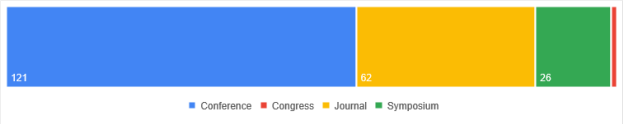


Figure 5: Venue distribution.

5.1.2 Frequency of Publication. Fig 6 summarizes the demographic data showing that the majority of the studies were conducted the most recent, 2014 onward. This rise of studies in the particular topic may be evidence for an increase in relevance for the software engineering research community.

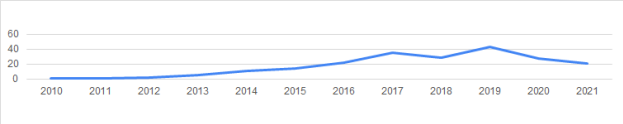


Figure 6: Distribution of selected studies over the analyzed period.

5.1.3 Study Origin. Fig 7 displays the country distribution for the accepted articles, showing a lead for China and USA. The others contain all countries with less than 10 studies found.

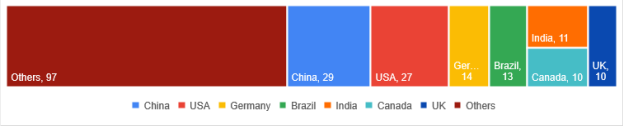


Figure 7: Studies divided by countries.

5.2 Research Questions

In this subsection, the extracted data is used to answer each one of the research questions, presented in section 3

5.2.1 RQ01: How often is every quality requirement cited in the analyzed articles. The studies were categorized based on the quality characteristics they discussed. These characteristics were either mentioned only briefly as a side effect of the study, or were a direct focus of the work being proposed.

In Figure 8, the quality characteristics by year are presented and it shows a significant increase for specifics characteristics such as Security and Performance. The quality characteristic classification was executed during the full article reading phase.

Whenever possible the studies were classified by the quality characteristics as defined by the ISO25010 ISO 25010:2011 [13]. However many studies did not use one of the ISO's standardized terms, leaving to the authors of this work to disambiguate and classify them, notable mentions include Quality of Service (QoS), which can bear different meanings, such when regarding the overall performance or to traffic prioritization in the networking context."Quality" refers to when the study did not specify which quality characteristic is being taken in consideration.

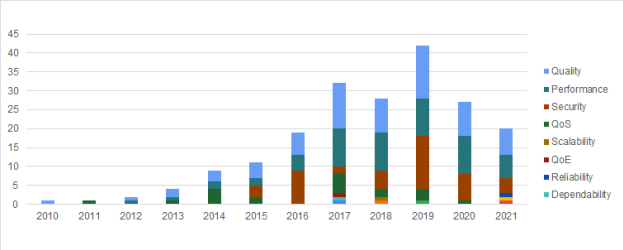


Figure 8: Quality characteristic by year.

It was possible to plot a bubble chart of quality characteristics against the study domain (Fig 9) (relations with less than 2 items per row/column were removed in order to maintain readability). We determined the study domain by the study's primary objective, for instance, if the study proposed an implementation focused on bettering communication from a client-side device, its domain is defined as "Communication". As expected the biggest correlations were where the study objective matched the mentioned quality characteristic. Less obvious information obtained from the chart was the distributed mention of performance.

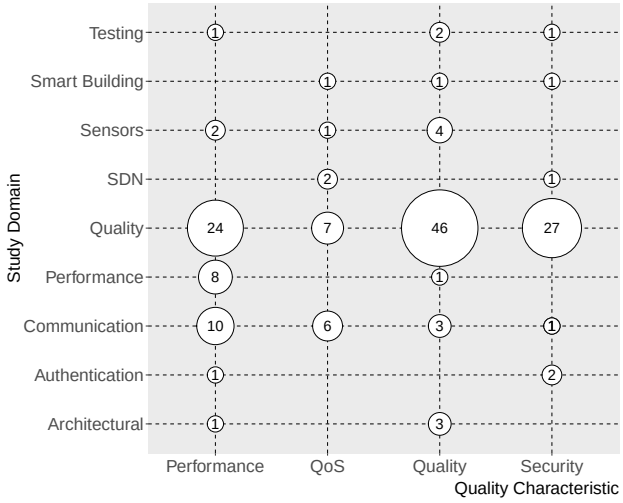


Figure 9: Study domain as contrasted by quality characteristics.

Answering this research question directly, 196 out of the 211 studies cited being conscious about a specific quality characteristic. Out of the 196, 45.92% of the studies had quality-related domains. The most popular characteristics were quality, security, performance, and QoS, all other quality characteristics were seldom found.

5.2.2 RQ02: Which software metrics, if any are being used for quality validation. For this section, the expectation was to be able to illustrate the popularity of specific metrics or paradigms. For example Average Method Lines of Code (AMLOC) or a collection of metrics for Object-Oriented Programming.

Although "software metric" and related terms were used in the search string, only 10 out of the 211 studies (<5%) cited the use of software metrics in their work. From these results, it would be fair to assume the absence of software metrics to quite a high degree.

Due to the expressive results, one may be instigated into researching the efficacy and relevancy of software metrics in modern contexts. In spite of its limitation and origin in the early 70s, software metrics are routinely revised into modern times, keeping its usefulness, such work as Padmini et al. [24] which focuses on agile development.

This research question is being answered by the mostly absent in the use of software metrics in the analyzed studies.

5.2.3 RQ03: What type of experiment is being used for research validation. In this subsection, we analyzed the class of the studies and the type of validations conducted by each.

Regarding the type of the studies, they were arranged into the five classes as proposed by Roberto et al. [27] (Fig 10), as expected only three of these were populated:

Figure 10: Study type.

- Action Research: consists of a study that attempt to solve a real-world problem while studying the experience of solving the problem. This class is the most popular was expected as most mapped studies are composed mostly by approach proposals.
- Case Study: In the context of software engineering is defined as an empirical inquiry, exploring and investigating phenomena with real-life context, does not have to be exactly defined.
- Controlled Experiments: consists of well-selected and manipulated variables in order to measure the effect caused in other variables.

From the results, in Fig 11 we can note that only 107 studies mentioned experiments for approach validation or testing. The most commonly used technique among the studies was internal testing, where the authors tested their own work to validate its functionality and quality. Only one study had external experimentation implemented by a third party, another one yet had an online public validation.

Figure 11: Validation type.

The retrieved data successfully answers this research question, finding out that the majority of studies did not inform the use of experiments, either because it was internally tested or not applicable.

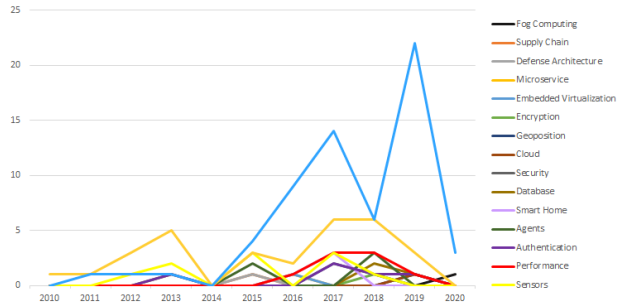


Figure 12: Studies' domain.

5.2.4 RQ04: Which platform, domain, framework, and/or paradigm is being used. This extraction has the most open-ended field, requiring a greater effort on filtering and grouping information. We gathered hardware and software platforms and frameworks, different communication techniques, database domains, module-specific solutions. It is possible to notice trends such as a growing focus on quality and security studies.

In Fig 12 they are grouped and represented by year, notable trends are the growing focus in quality and security studies.

It was possible to chart the popularity of the study domain by programming language, most used being C/C++ and Java (Fig 13), only 78 out of the 211 studies mentioned the programming language used. Another topic for discussion was the absence of a single study specifying the programming paradigm. Programming paradigms can have an effect on software quality, Objected Oriented Programming (OOP) capable languages may have not been used to their full potential.

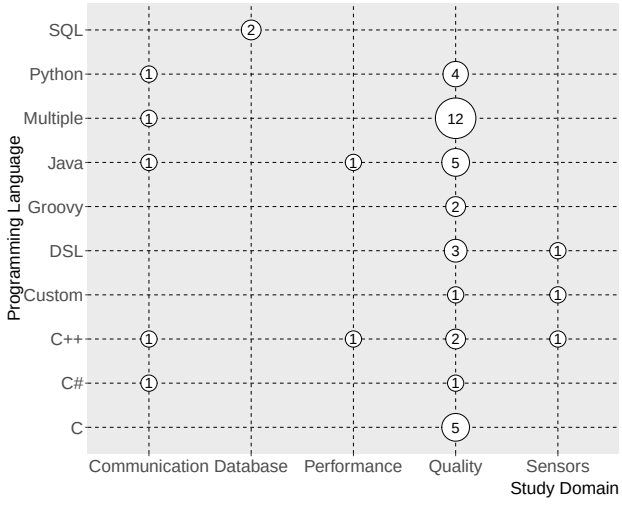


Figure 13: Programming language as contrasted by study domain.

Development platforms were barely cited as compared to programming language discussed previously, 14 shows the popularity as contrasted with a programming language (relations with less than 2 values were ignored for better readability). Azure, AWS, and SIoT were the most popular after the prototyping platforms such as Arduino and Raspberry Pi. Further filtering of the data identified a clear trend of rapidly increased adoption of these prototyping platforms after their appearance on the first study in 2015.

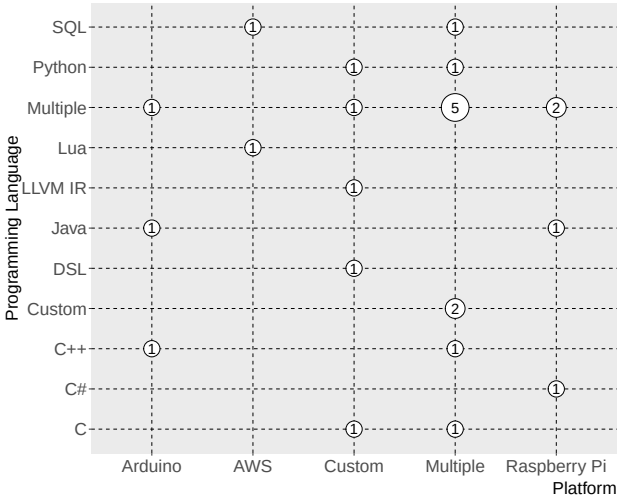


Figure 14: Programming language as contrasted by platform.

6 DISCUSSION

In this section, the results, threats to validity, and future works are discussed.

The growing concern with quality is evidenced in the mapping results, however, the quality concerns are more directly related to user experience and product performance, software quality is seldom cited. The lack of software metrics also showcases the lack of concern with software quality.

An observation to be made when regarding programming paradigms is the following, all the specified programming languages are multi-paradigm while no paradigm was cited. 66 studies specified using programming languages that support object-oriented programming (OOP), despite it, none study mentioned the paradigm itself or the concern of complying with a certain design pattern or methodology.

Only 107 out of the 211 studies cited the use of validation of the proposed solution, and from those 94 were internal tests, carried out by the authors themselves

Additional correlations can be made when addressing programming languages collected in the systematic mapping, Fig 15 shows programming language as contrasted by the platform while Fig 16 is contrasted by quality characteristic.

6.1 Threats to Validity

The results in this systematic mapping may be affected by the incompleteness of search, process bias, and inaccuracy in any process execution. The search string is the main threat to excluding potentially relevant studies by excluding synonyms to the used terms, Relevant studies can potentially be missing from publishers that were not included in the mapping. Despite the high Kappa

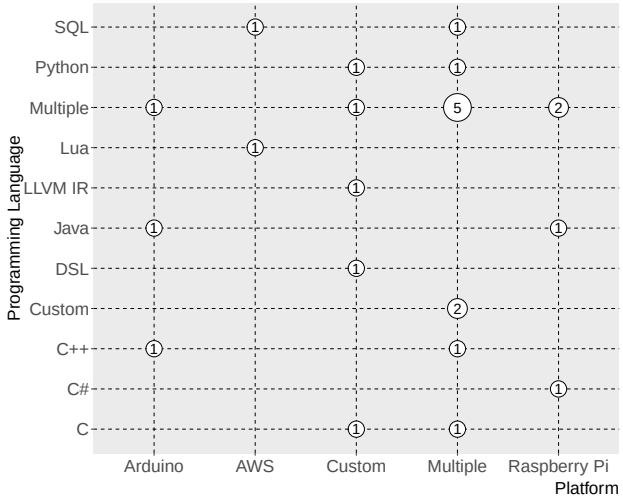


Figure 15: Programming language as contrasted by platform.

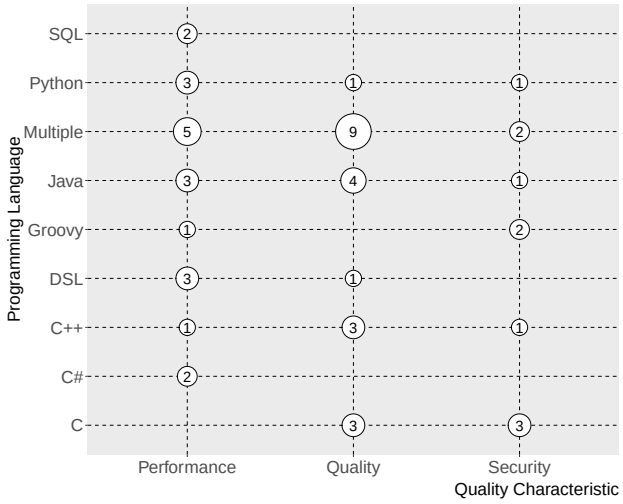


Figure 16: Programming language as contrasted by quality characteristics.

coefficient value, author bias may have affected the manual data extraction process.

6.2 Future Works

This work lays the foundation for multiple future studies, by focusing on specific factors raised in this original systematic mapping, for example:

- Kuri et al. [18] produced a mapping study that seeks a better understanding of software quality metrics in the Virtual Reality paradigm, categorizing the paradigm in themes, something that is yet to be done in IoT.
- Padmini et al. [24] did a study on the use of software metrics in agile software development, studying the applicability of some particular metrics to the agile domain, contrasting to the Traditional Software Development process. Likewise every single metric should be analyzed and validated inside the IoT domain.
- Some works such as the one produced by Li [20] are a staple of software metrics, where the author applies Kitchenham's framework for validating software metrics in order to find deficiencies and propose a new suite/set of metrics for a specific domain, in this case, Object-Oriented Programming.

Future works may also target resolving the treats to validity specified in this section whenever applicable.

7 CONCLUSION

In the systematic mapping, it was executed searches in major publications for relevant studies in IoT's software quality. This study presents evidence of an increase in IoT publications, nevertheless, it was identified many gaps in the IoT client-side development.

We did not find any study using software metrics to analyze the code quality, or the paradigm of development used. It indicates the IoT software development of the client-side is still incipient, and developers are little concerned about software development best practices.

Furthermore, the IoT client-side development is very much related to physical components, and these components may be replaced by similar or may use in different situations. Therefore, modularization, reuse, and maintainability should be a constant concern in this kind of development. However, considering the results found, the studies did not focus on these aspects. Other studies already addressed the problem of the poor quality of IoT software development on the client-side, such as Van Oorschot and Smith which describes the real world consequences of security problems in the IoT client-side development and Bures et al. which relates performance problems and quality challenges.

Thus, the main contribution of this study is to show how IoT client-side development presents many software engineering and development gaps. Studies on techniques and tools to improve software development must be addressed in further research.

REFERENCES

[1] [n. d.]. World Internet Users Statistics and 2019 World Population Stats. https://www.internetworldstats.com/stats.htm

[2] Fadele Ayotunde Alaba, Mazliza Othman, Ibrahim Abaker Targio Hashem, and Faiz Alotaibi. 2017. Internet of Things security: A survey. *Journal of Network and Computer Applications* 88, March (2017), 10–28. https://doi.org/10.1016/j.jnca.2017.04.002

[3] Mohsen Hallaj Asghar, Atul Negi, and Nasibeh Mohammadzadeh. 2015. Principle application and vision in Internet of Things (IoT). *International Conference on Computing, Communication and Automation, ICCCA 2015* (2015), 427–431. https://doi.org/10.1109/CCAA.2015.7148413

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (oct 2010), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

[5] Miroslav Bures, Tomas Cerny, and Bestoun S. Ahmed. 2018. Internet of Things: Current challenges in the quality assurance and testing methods. *arXiv* (2018), 625–634.

[6] Louis Coetzee and Johan Eksteen. 2011. Internet of things–promise for the future? An Introduction. (2011).

[7] Mariela Cortés, Raphael Saraiva, Marcia Souza, Patricia Mello, and Pamella Soares. 2019. Adoption of software testing in internet of things: a systematic literature mapping. In *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*.

[8] Ricardo Theis Geraldi, Sheila Reinehr, and Andreia Malucelli. 2020. Software product line applied to the internet of things: A systematic literature review.

Information and Software Technology 124, February (2020), 106293. https://doi.org/10.1016/j.infsof.2020.106293

[9] Richard Grimmett. 2015. *Raspberry Pi robotics projects.* Packt Publishing Ltd.

[10] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[11] Jenalea Howell. [n. d.]. Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says. https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says

[12] Mehdi Imani, Abolfazl Qiasi Moghadam, Nasrin Zarif, Maaruf Ali, Omekolsoom Noshiri, Kimia Faramarzi, Hamid Arabnia, and Majid Joudaki. 2018. A comprehensive survey on addressing methods in the Internet of Things. *arXiv preprint arXiv:1807.02173* (2018).

[13] ISO 25010:2011 2011. *ISO25010 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* Standard. International Organization for Standardization.

[14] Adeel Javed. 2016. *Building Arduino projects for the Internet of Things: experiments with real-world applications.* Apress.

[15] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology* 51, 1 (2009), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

[16] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. 2010. The value of mapping studies – A participant-observer case study. (2010). https://doi.org/10.14236/ewic/ease2010.4

[17] Hari Kishan Kondaveeti, Nandeesh Kumar Kumaravelu, Sunny Dayal Vanambathina, Sudha Ellison Mathe, and Suseela Vappangi. 2021. A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations. *Computer Science Review* 40 (2021), 100364. https://doi.org/10.1016/j.cosrev.2021.100364

[18] Mohit Kuri, Sai Anirudh Karre, and Y Raghu Reddy. 2021. Understanding Software Quality Metrics for Virtual Reality Products - A Mapping Study. In *14th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference) (ISEC 2021).* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3452383.3452391

[19] JR Landis and GG Koch. 1977. Landis amd Koch1977_agreement of categorical data. *Biometrics* 33, 1 (1977), 159–174.

[20] Wei Li. 1998. Another metric suite for object-oriented programming. *Journal of Systems and Software* 44, 2 (1998), 155–162. https://doi.org/10.1016/S0164-1212(98)10052-3

[21] Leonardo Ribeiro Machado, Francisco J da Silva, Alex Barradas, Davi Viana, Ariel Teles, and Luciano Coutinho. 2020. Product Quality for Smart Cities Applications: A Mapping Study. In *XVI Brazilian Symposium on Information Systems.* 1–8.

[22] Phu Hong Nguyen, Nicolas Ferry, Gencer Erdogan, Hui Song, Stéphane Lavirotte, Jean-Yves Tigli, and Arnor Solberg. 2019. A Systematic Mapping Study of Deployment and Orchestration Approaches for IoT.. In *IoTBDS.* 69–82.

[23] Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, and Carlos Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *Journal of Systems and Software* 128 (2017), 164–197. https://doi.org/10.1016/j.jss.2017.03.044

[24] K. V.Jeeva Padmini, H. M.N. Dilum Bandara, and Indika Perera. 2015. Use of software metrics in agile software development process. *MERCon 2015 - Moratuwa Engineering Research Conference* (2015), 312–317. https://doi.org/10.1109/MERCon.2015.7112365

[25] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12.* 1–10.

[26] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[27] Rafael Roberto, João Paulo Lima, and Veronica Teichrieb. 2016. Tracking for mobile devices: A systematic mapping study. *Computers and Graphics (Pergamon)* 56 (2016), 20–30. https://doi.org/10.1016/j.cag.2016.02.002

[28] Paul C. Van Oorschot and Sean W. Smith. 2019. The Internet of Things: Security Challenges. *IEEE Security and Privacy* 17, 5 (2019), 7–9. https://doi.org/10.1109/MSEC.2019.2925918

[29] Gary White, Vivek Nallur, and Siobhán Clarke. 2017. Quality of service approaches in IoT: A systematic mapping. *Journal of Systems and Software* 132 (2017), 186–203.

[30] Sabine Wolny, Alexandra Mazak, and Bernhard Wally. 2018. An Initial Mapping Study on MDE4IoT.. In *MODELS Workshops.* 524–529.

[31] Michalis Xenos, D Stavrinoudis, K Zikouli, and D Christodoulakis. 2000. Object-oriented metrics-a survey. *Proceedings of the FESMA*, 1–10.

PUBLISHED WORKS BY THE AUTHOR

Published Works by the Author during the program.

1. Rafael Figueira Goncalves, André Luís Andrade Menolli, **MDD4IoT: Model-Driven Development Framework Proposal for Internet of Things**, SBQS 2021 - WTDQS, article 18554

2. Rafael Figueira Goncalves, André Luís Andrade Menolli, Gustavo Dionisio, **Systematic Mapping on Internet of Things' Client-Sided Development**, SBSI 2022 - Main Track, 10.1145/3535511.3535541

3. Rafael Figueira Goncalves, André Luís Andrade Menolli, Gustavo Dionisio, **MDD4CPD: Model Driven Development Approach Proposal for Cyber-Physical Devices**, SBSI 2022 - Main Track, 10.1145/3535511.3535542