Journal of Universal Computer Science, vol. 32, no. 11 (2026), 1612-1649
submitted: 14/8/2025, accepted: 5/2/2026, appeared: 28/10/2026 *CC BY 4.0*

Model of Things (MoT): A Model-Driven Approach to Streamline Cloud-of-Things Application Development

Cristiano Welter
(PPGCA, University of Vale do Rio dos Sinos, São Leopoldo, RS, Brazil
cristianowelter@gmail.com)

Kleinner Farias
(PPGCA, University of Vale do Rio dos Sinos, São Leopoldo, RS, Brazil
https://orcid.org/0000-0003-1891-3580, kleinnerfarias@unisinos.br)

Abstract: The convergence of Cloud Computing and the Internet of Things (IoT) underpins modern, scalable, and data-intensive systems. Cloud-of-Things (CoT) applications are now pervasive across healthcare, smart cities, and industrial automation. Nevertheless, their development remains fragmented and technically demanding. Current CoT development approaches require substantial expertise in heterogeneous platforms, protocols, and deployment technologies. They offer limited model-driven support, weak automation, and poor alignment with low-code principles. As a result, developing CoT applications remains effort-intensive and error-prone. This study aims to simplify CoT application development by introducing the Model of Things (MoT). MoT integrates UML-based modeling with low-code execution concepts to reduce technical barriers while preserving automation and deployment feasibility. We designed a custom UML profile comprising nine CoT-specific stereotypes and an automated transformation pipeline. MoT was evaluated using one end-to-end case study and a Technology Acceptance Model (TAM) questionnaire with 12 participants with software modeling experience. In the evaluated case study, MoT generated executable application components from UML models, while users supplied the required configuration parameters and deployment settings. TAM results indicate strong acceptance: all 12 participants rated MoT positively for usefulness and ease of use, and qualitative feedback emphasized reduced complexity and faster development. MoT demonstrates that combining UML modeling with low-code execution is feasible and valuable for CoT systems. By enabling executable models and lowering entry barriers, MoT advances practical, model-driven CoT engineering and opens avenues for broader industrial adoption.

Keywords: Model of Things, UML, Low-code, Model-based Development
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1
DOI: 10.3897/jucs.168743

1 Introduction

The *Cloud of Things* (CoT) paradigm emerges from the convergence of Cloud Computing and the Internet of Things (IoT), enabling scalable processing, storage, and orchestration of data generated by interconnected devices [Suhailam et al. 2024, Aazam et al. 2016]. CoT systems increasingly support critical domains such as healthcare monitoring, smart cities, industrial automation, and intelligent environments [Alhaidari et al. 2023, Yadav et al. 2024]. As IoT deployments scale and become more heterogeneous, cloud infrastructure becomes essential for handling computation-intensive tasks, real-time analytics,

and elastic resource management. Despite their importance, the CoT applications remain difficult to engineer, integrate, and evolve due to the diversity of devices, communication protocols, cloud services, and deployment configurations [Gubbi et al. 2013]. The demand for rapid development cycles and continuous evolution amplifies these challenges. Consequently, there is a strong need for approaches that raise the level of abstraction, promote automation, and reduce the technical expertise required to build and deploy CoT systems. Model-driven development and low-code paradigms have been identified as promising means to address these needs by enabling developers to focus on system intent rather than platform-specific implementation details [Alhaidari et al. 2023]. However, their integration within CoT engineering remains limited and fragmented.

Although previous studies have explored formal modeling, model-driven engineering, and deployment automation for IoT and CoT systems [Hellal et al. 2024, Belguidoum et al. 2022, de Oliveira and Lopes 2020], essential gaps *persist*. Formal approaches often emphasize verification and correctness but offer limited accessibility for practitioners and non-expert developers [Hellal et al. 2024]. Conversely, deployment- and automation-oriented solutions prioritize execution efficiency while relying on low-level configuration artifacts, such as scripts or declarative descriptors, that bypass high-level modeling abstractions [Hassan and Qasha 2021]. Low-code platforms reduce development effort but frequently lack explicit model-based foundations, making reasoning, evolution, and consistency management difficult [Panayiotou et al. 2025]. Moreover, existing UML-based profiles and domain-specific languages for IoT often stop at code generation or simulation, without establishing a direct, executable link between modeling constructs and cloud-ready deployment artifacts [Fazel and Wainer 2023, Fortas et al. 2022a]. As discussed in Section 3, the current literature still lacks approaches that simultaneously integrate UML modeling, low-code execution, and automated deployment in a coherent, tool-supported pipeline. This gap limits the practical adoption of model-driven techniques in real-world CoT development.

To address these limitations, this article introduces the *Model of Things* (MoT), a tool-supported, model-based approach designed to streamline the development of Cloud-of-Things applications. MoT aims to reduce technical barriers while preserving automation and executability by combining UML-based modeling with low-code execution principles. MoT operationalizes this objective through three core elements: a custom UML profile that captures CoT-specific concerns, an automated transformation pipeline that maps models to executable artifacts, and a deployment infrastructure that supports cloud-based execution. To assess the approach's feasibility and practical value, we used a qualitative research method aligned with exploratory and engineering-oriented studies [Wohlin et al. 2012]. We conducted an end-to-end case study demonstrating model-to-execution transformation and used a Technology Acceptance Model (TAM) questionnaire to capture user perceptions of usefulness and ease of use [Davis 1989, Marangunić & Granić 2015]. This method allows us to assess whether MoT, as designed, is understandable, usable, and meaningful to developers with experience in software modeling and CoT-related technologies.

Our initial results indicate that MoT successfully enables the transformation of UML models into executable CoT applications without requiring manual coding or low-level deployment configuration. In our case study, the proposed toolchain generated and deployed the complete application, demonstrating end-to-end executability. Our TAM-based evaluation involved 12 participants with backgrounds in software modeling and development. All participants reported positive perceptions regarding MoT's usefulness and ease of use, indicating strong acceptance of the approach. Participants perceived MoT as potentially useful for reducing development complexity and delivery effort; however,

these benefits were not objectively measured. Qualitative feedback emphasized reduced perceived complexity, more precise separation of concerns, and faster development compared to traditional CoT workflows. The novelty of these results lies not in algorithmic optimization but in demonstrating that executable CoT systems can be derived directly from UML models enriched with low-code semantics, unlike existing approaches that either remain conceptual or focus solely on automation scripts. MoT bridges modeling abstraction and practical deployment, showing that model-driven and low-code paradigms can coexist in a coherent engineering solution.

This work makes three main contributions. First, it introduces MoT, an engineering-focused, model-based approach that integrates a custom UML profile with automated, low-code transformations tailored to developing CoT applications. This contribution benefits researchers and practitioners seeking executable model-driven solutions beyond conceptual modeling. Second, it presents a UML profile extension that encapsulates IoT and cloud concerns as first-class modeling elements, enabling consistent, automated transformations while reducing the expertise required to build CoT systems. This contribution directly supports software developers and system architects by improving clarity, reuse, and consistency. Third, the study provides empirical evidence through a case study and user assessment, offering qualitative insights into feasibility, perceived usefulness, and ease of integration. These results benefit the broader research community by clarifying the practical limits and potential of low-code, model-driven CoT development. These contributions can lay a foundation for future work on scalability, performance evaluation, and cross-platform generalization, while delivering an immediately usable, executable approach for CoT engineering.

The remainder of this article is organized as follows. Section 2 explains the main concepts used in our study. Section 3 contrasts our study with the current literature. Section 4 presents the proposed approach. Section 5 outlines how we evaluated the proposed approach. Section 6 presents some conclusions and future directions.

2 Background

This section introduces two fundamental concepts: Section 2.1 provides an overview of Cloud of Things (CoT), while Section 2.2 discusses Node-Red and its role in facilitating IoT application development.

2.1 Cloud of Things

The Cloud of Things (CoT) is a paradigm integrating the Internet of Things (IoT) with cloud computing, enabling scalable, efficient management of data and resources from diverse IoT devices [Alhaidari et al. 2023, Yadav et al. 2024, Aazam et al. 2016]. By connecting resource-constrained devices, such as sensors, to cloud infrastructure, CoT enables the processing, analysis, and storage of large volumes of data generated by IoT. This integration provides a flexible, resilient environment that enhances the performance, reliability, and adaptability of IoT applications, transforming raw data into valuable, actionable services.

Investigating CoT in model-based software development is essential for three reasons. First, model-driven approaches can streamline the development and deployment of IoT applications on cloud platforms. CoT applications can be deployed and scaled automatically by abstracting complex cloud configurations into high-level models. This reduces human error and optimizes resource use. Second, model-based development

enables faster adaptation to changing requirements, a critical need for IoT systems that often integrate diverse, evolving device types and data streams. In this way, developers can quickly modify and redeploy CoT applications. For example, this can help them stay compatible with new devices and standards. Third, UML models can help to improve consistency across CoT system components, simplifying maintenance and updates. This approach fosters maintainable architectures in which changes to IoT devices or cloud services are quickly reflected in a unified, model-driven design, improving system longevity and reducing costs.

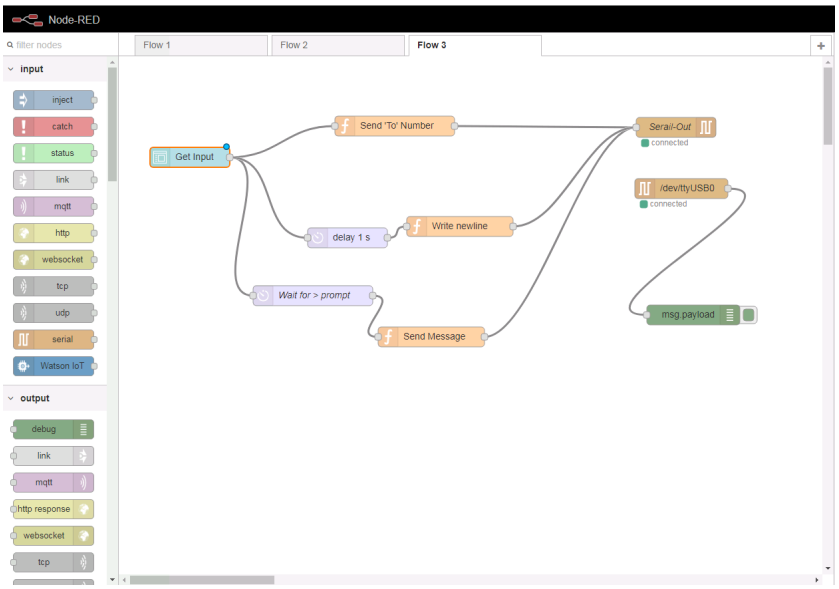


Figure 1: A Node-Red interface that shows an illustrative Node-Red workflow using typical input and output components.

2.2 Node-Red

Node-Red[1], illustrated in Figure 1, is a low-code development tool designed for building event-driven applications by integrating hardware, APIs, and online services. It offers a browser-based flow editor that lets users configure application logic via a simple drag-and-drop interface. Built on Node.js, Node-RED's lightweight, event-driven runtime is optimized for low-cost devices like the Raspberry Pi and scalable cloud environments. With its extensive library of nodes and compatibility with over 225,000 Node.js modules, Node-RED simplifies complex integrations and supports rapid deployment of IoT and web-based applications.

We adopted Node-Red in our proposed MoT approach for two reasons. First, Node-Red's low-code interface enables rapid integration of diverse IoT devices and data sources.

[1] Node-Red: https://nodered.org/

This ease of use streamlines the process of connecting sensors and cloud services within the MoT framework. We believe this can promote data flow management and reduce the need for extensive coding. Secondly, the event-driven model built on Node.js supports real-time data processing, which is essential for CoT applications. Its compatibility with a wide range of APIs and services also enhances the MoT approach's adaptability, enabling seamless scalability and responsiveness in handling IoT-driven events.

3 Related Work

Research on CoT applications has evolved along several interconnected strands that collectively shape the field's current state. These include: (i) *formal modeling and verification approaches* that ensure behavioral correctness of IoT–Cloud systems, (ii) *dataflow-centric and decentralized execution frameworks* designed to manage large-scale and distributed IoT data processing, (iii) *model-driven engineering (MDE) solutions* that promote abstraction, automation, and systematic code generation, and (iv) *deployment-oriented frameworks* that aim to reduce configuration burden and operational complexity in heterogeneous CoT environments. These lines of research are mature, yet each reflects distinct priorities and trade-offs.

Despite these advances, most existing approaches privilege either *formal rigor over accessibility* or *automation over structured model-driven design*. Formal solutions often provide strong guarantees but demand specialized expertise, limiting their applicability to broader developer communities. Conversely, automation-oriented and execution-focused frameworks streamline deployment but rarely treat modeling semantics as a first-class concern. Moreover, few works objectively integrate *UML-based modeling*, low-code development principles, and executable automation in a unified, coherent manner. To analyze the literature rigorously and fairly, our review adopts explicit criteria, i.e., modeling paradigm, research focus, automation level, deployment support, and target users, enabling objective comparison rather than subjective assessment.

To this end, we present the procedures adopted to select our related studies (Section 3.1), analyze each selected study (Section 3.2), and then present a comparative analysis and research opportunities (Section

3.1 Study Selection Procedures

The selection of related studies followed a structured process to ensure relevance, rigor, and alignment with this research's objectives. First, the authors' prior research experience in the Cloud-of-Things, Internet of Things, and Model-driven Engineering domains guided the identification of seminal and representative contributions. Second, a structured search strategy was applied using the string *((''Cloud of Things'' OR ''Internet of Things'') AND UML AND ''Low code'' AND ''Model-based Development'')*. This query was executed in two widely recognized digital libraries—Google Scholar[2] and IEEE[3]—to capture both academic impact and breadth of coverage. Third, only studies directly addressing themes aligned with our research focus were retained, particularly those that explored model-based development, automation, abstraction, and the feasibility of deployment in IoT or CoT systems. Works that merely cited these domains without engaging their core challenges were discarded. This three-step procedure ensured that

[2] Google Scholar: https://scholar.google.com/
[3] IEEE Xplore: https://ieeexplore.ieee.org/Xplore/home.jsp

the selected works address a research problem closely related to that investigated in this study and thus provide a meaningful basis for comparison.

This selection strategy is rigorous for three main reasons. First, it follows established methodological practices used and validated in prior peer-reviewed software engineering and IoT research, ensuring methodological soundness. Second, we carefully constructed the search string to reflect the core dimensions of our investigation (i.e., Cloud/IoT context, UML and model-based development, and low-code principles), reducing ambiguity and minimizing irrelevant results. Third, we focused on contributions published between 2020 and 2026, prioritizing work that reflects the latest advancements and technological constraints in the CoT ecosystem. We excluded literature reviews, surveys, and systematic mappings to focus exclusively on primary, solution-oriented contributions. Together, these procedures ensure the selected corpus is representative and tightly aligned with the research goals.

3.2 Analysis of Related Work

Modeling paradigm. Existing studies employ diverse modeling strategies to address the complexity of Cloud–IoT ecosystems. Hellal et al. [Hellal et al. 2024] adopt Event-B to formally specify and verify the structural, functional, operational, and behavioral properties of IoT–Cloud systems. In contrast, Fazel and Wainer [Fazel and Wainer 2023] employ the Discrete Event System Specification (DEVS) formalism to support model-driven simulation and analysis of IoT systems. Other studies emphasize dataflow modeling, as Zenoh-Flow [Baldoni et al. 2023], organizing ML-powered IoT applications into decentralized data pipelines. Model-Driven Development (MDD) also underpins microservice-oriented modeling [Belguidoum et al. 2022], ThingML-based development [Fortas et al. 2022a], WoT-based frameworks [de Oliveira and Lopes 2020], and cross-platform IoT domain-specific modeling languages (DSMLs) [Corradini et al. 2023]. More recently, LocSys [Panayiotou et al. 2025] has introduced a multi-DSL approach to support complex cyber-physical systems (CPS). Although these studies advance modeling abstraction and automation, they pursue different objectives, including formal verification, simulation, interoperability, and platform-independent development. Few explicitly integrate UML-based modeling, low-code execution, and automated deployment into a unified workflow accessible to developers with varying levels of expertise, which motivates the MoT approach.

Research focus. The literature addresses several distinct but complementary concerns. Works such as [Hellal et al. 2024, Fazel and Wainer 2023] address correctness, verification, and behavioral assurance. Others prioritize system management, ML processing, and performance optimization in distributed IoT pipelines, as in [Baldoni et al. 2023]. Security-driven models are emphasized in [Sree et al. 2022], while architectural scalability and microservices structuring guide [Belguidoum et al. 2022]. [Hassan and Qasha 2021, Terracher et al. 2024] explicitly target deployment and distributed execution simplification, and [de Oliveira and Lopes 2020]. Meanwhile, advanced optimization and networking intelligence dominate [Jiuting and Mohajer 2025, Mohajer et al. 2024, Kuang et al. 2025]. LocSys [Panayiotou et al. 2025] and X-IoT [Corradini et al. 2023] enhance usability and portability by providing refined platform-independent abstractions. However, despite this breadth, most works do not explicitly seek to reduce development barriers for non-expert developers via low-code principles integrated with MDD, nor do they explicitly couple modeling semantics with executable, cloud-ready structures as MoT does.

Automation level. Automation appears with distinct emphases across studies. Full formal verification pipelines, as in [Hellal et al. 2024], focus on correctness rather than execution automation. Automated ML data pipelines emerge in [Baldoni et al. 2023], while simulation-driven automatic execution is supported in [Fazel and Wainer 2023]. Microservice generation is automated in [Belguidoum et al. 2022], and ThingML [Fortas et al. 2022a] automates multi-platform code generation. Cross-platform deployment automation appears in [Hassan and Qasha 2021] and [Corradini et al. 2023]. However, few tools automate end-to-end transformation from conceptual UML modeling to low-code executable artifacts, maintaining semantic continuity across steps. MoT explicitly addresses this blind spot by providing transparent transformation and gradual automation aligned with CoT deployment.

Deployment support. Deployment capabilities vary markedly. Simulation-oriented approaches [Fazel and Wainer 2023, Fortas et al. 2022a] emphasize experimentation but require further effort for operationalization. Cloud deployment is explicit in [Hellal et al. 2024, de Oliveira and Lopes 2020, Hassan and Qasha 2021, Terracher et al. 2024], while decentralized runtime support dominates in [Baldoni et al. 2023]. Performance-centric cloud execution is the primary concern of [Jiuting and Mohajer 2025, Mohajer et al. 2024, Kuang et al. 2025]. LocSys [Panayiotou et al. 2025] provides execution pipelines but focuses on CPS rather than CoT execution. Although these studies advance cloud execution, most either assume expert users or introduce significant configuration complexity. MoT bridges this gap by explicitly connecting modeling constructs to deployable components using a structured, low-code execution backbone.

Target users. Most reviewed approaches implicitly assume experienced developers, formal method specialists, or IoT engineers. Event-B [Hellal et al. 2024] and DEVS simulation [Fazel and Wainer 2023] demand specialized expertise. ML-centric frameworks [Baldoni et al. 2023] and networking optimization works [Jiuting and Mohajer 2025, Mohajer et al. 2024, Kuang et al. 2025] are intended for advanced researchers and engineers. Microservice- and WoT-oriented solutions [Belguidoum et al. 2022, de Oliveira and Lopes 2020, Hassan and Qasha 2021] still require architectural understanding. Only a few explicitly prioritize accessibility and usability, notably LocSys [Panayiotou et al. 2025], which demonstrates strong user acceptance. Yet even these solutions rarely provide a streamlined, low-code path from UML modeling to concrete CoT deployment, which MoT aims to achieve.

In summary, the current literature shows maturity in modeling rigor, architectural design, and performance optimization, but it lacks comprehensive, low-code, model-driven solutions that explicitly reduce development barriers, preserve semantic continuity from models to execution, and support CoT deployment with accessibility as a first-class concern. These gaps define the core contribution space of our proposed approach.

3.3 Comparative Analysis and Research Gaps

Table 1 presents our comparative analysis, drawing directly on the objective gaps identified rather than subjective perceptions. The synthesis of the selected works shows ongoing evolution in Cloud-of-Things engineering. However, it also reveals structural limitations that constrain development accessibility, architectural clarity, and execution feasibility. By analyzing studies against criteria such as modeling paradigm, research focus, automation level, deployment support, and target users, we identify three persistent research gaps that the current literature has not sufficiently addressed.

First, there is limited integration between UML-based modeling and low-code execution environments. Several solutions emphasize strong modeling capabilities [Hellal

Work	Modeling Paradigm	Research Focus	Automation Level	Deployment Support	Target Users
MoT approach	*UML modeling*	*Automation + Streamline*	*High*	*Model-oriented + Cloud Agnostic*	*Non-experts*
[Hellal et al. 2024]	Formal/Event-B	Verification	Medium	Cloud-oriented	Experts
[Baldoni et al. 2023]	Dataflow Framework	ML Pipelines	High	Decentralized IoT	Developers
[Fazel and Wainer 2023]	DEVS MDD	Simulation + Design	High	Simulation + Runtime	Experts
[Sree et al. 2022]	Security Model	Secure Access	Low	Cloud	Practitioners
[Belguidoum et al. 2022]	SysML/MDD	Microservices	Medium	IoT Runtime	Developers
[de Oliveira and Lopes 2020]	MDE/WoT	Interoperability	Medium	Cloud/WoT	Developers
[Fortas et al. 2022a]	DSL (ThingML)	Code Gen + Testing	High	Multi-platform	Developers
[Hassan and Qasha 2021]	Declarative YAML	Distributed IoT Deploy	Medium	Cloud + Edge	DevOps Engineers
[Terracher et al. 2024]	MDE	Sensor Data Exploitation	Medium	Cloud	Engineers
[Jiuting and Mohajer 2025]	Optimization Model	Network Performance	High	Satellite Cloud	Researchers
[Mohajer et al. 2024]	DRL Framework	Edge Offloading	High	Edge Cloud	Researchers
[Kuang et al. 2025]	MADL/GNN	Reliability + QoS	High	MANET Cloud	Researchers
[Panayiotou et al. 2025]	Multi-DSL MDD	CPS Usability	Medium	Runtime Pipelines	Mixed Users
[Corradini et al. 2023]	DSML MDD	Cross-Platform IoT	Medium	Platform-Independent Deploy	Developers

Table 1: A comparative analysis of the selected studies.

et al. 2024, Fazel and Wainer 2023, Belguidoum et al. 2022, Corradini et al. 2023], while others privilege execution, deployment, and runtime efficiency [Baldoni et al. 2023, Fortas et al. 2022a, Hassan and Qasha 2021]. Yet, few approaches bridge these two perspectives to enable non-expert or mixed-expertise users to move seamlessly from conceptual models to executable CoT applications. Second, many works excel either in formal rigor and theoretical soundness [Hellal et al. 2024, Fazel and Wainer 2023] or in automation and optimization [Jiuting and Mohajer 2025, Mohajer et al. 2024, Kuang et al. 2025], but rarely balance abstraction with practical deployment capability, limiting their accessibility as engineering tools. Third, most solutions lack explicit end-to-end transformation pipelines that connect modeling constructs to deployable artifacts in a reproducible and extensible manner, particularly when heterogeneous cloud services and IoT platforms are involved.

The proposed MoT approach is designed to address these gaps. First, MoT introduces an engineering-focused solution centered on a UML profile tailored to CoT concerns, enabling structured design while maintaining familiarity for practitioners. Second, it defines a clear software architecture and implements a concrete tool-chain that automatically transforms models into platform-ready execution artifacts grounded in low-code principles. This moves beyond theoretical modeling by demonstrating operational feasibility. Third, MoT emphasizes practical validation. Beyond conceptual soundness, it is evaluated through a case study and user-centered assessment, providing qualitative evidence of usability, feasibility, and perceived usefulness, an aspect seldom addressed in comparable studies.

4 The Proposed Approach

This section presents *MoT (Model of Things*), a tool-supported, model-based approach that streamlines the development of Cloud-of-Things applications through low-code principles and UML model transformations. MoT emphasizes practicality and executability by tightly integrating modeling constructs with automated transformations, enabling a coherent path from design to deployment. We first provide an overview of the approach (Section 4.1), then introduce the proposed UML profile (Section 4.2) and summarize MoT's key features (Section 4.3). Next, we explain how model transformations are executed (Section 4.4) and describe the approach's underlying architecture (Section 4.5). We also discuss key implementation aspects (Section 4.6) and conclude by outlining the study's limitations (Section 4.7).

4.1 Overview of the Proposed Approach

Figure 2 presents an overview of our proposed 7-step process for streamlining CoT application development using low-code development and UML model transformations. The process comprises seven steps, each with specific inputs and outputs. Each step of this process is described as follows:

Step 1: Application modeling. This step aims at defining and specifying the requirements of CoT applications. Figure 3 shows the application modeling process. The application requirements are determined by creating UML use cases, a typical computation-independent model in model-based software development. The UML use cases can be created and refined as needed. It is an iterative process in which requirements are identified and refined through discussions among business specialists and technology experts. A use-case model shows how users (namely, actors) interact with the features of CoT

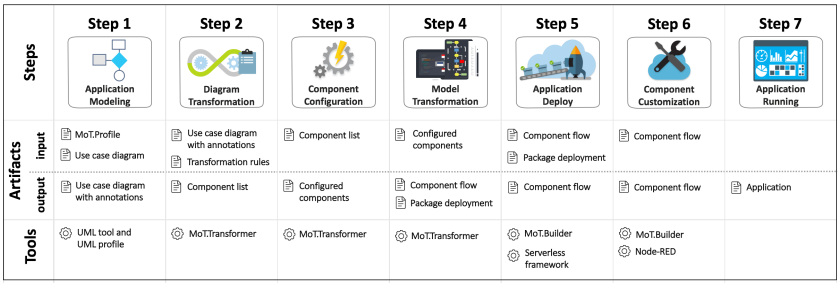


Figure 2: An overview of the MoT process. It presents a seven-step, model-based software development process, from modeling and transformation to deployment and execution, using specific artifacts and tools to ensure a structured, efficient workflow.

applications, with application features, and with other actors. In this sense, the MoT process requires essential inputs, such as use cases, actors, and their relationships. The MoT users apply the stereotypes of the MoT profile (described in Section 4.2) to customize the created use cases for the CoT domain. An annotated use case diagram is obtained from the use case diagram, with semantic information represented using stereotypes from the MoT profile. This model is then defined in XMI code, an OMG-standard XML-based information exchange format widely used by UML modeling tools. The XML code will be used as input for the next step.

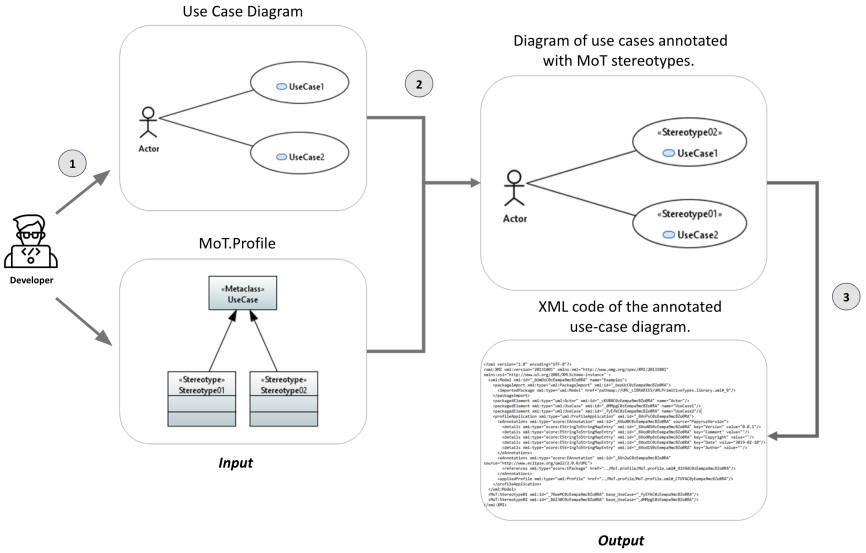


Figure 3: An overview of the supported modeling process. Developers create UML use case diagrams (Step 1) and apply MoT stereotypes manually (Step 2). After the MoT approach generates an XML code of the annotated diagram (Step 3).

Step 2: Diagram transformation. With the application model developed in the previous step and represented in XMI format, this step initiates the first transformation of models in the MoT approach. The MoT.Transformer maps all the diagram elements and identifies stereotypes compatible with the UML profile. It then transforms these UML elements into abstract application components. Figure 4 illustrates this transformation process. The elements in the use-case diagram, common diagram elements, and the *MoT.Profile* elements used in the diagram are all mapped and analyzed. The key player in this process is the model transformation tool, which contains a set of transformation rules used to generate the application components. These application components represent specific elements and features of a CoT application. Some of these components may require additional configurations not covered in the use-case diagram.

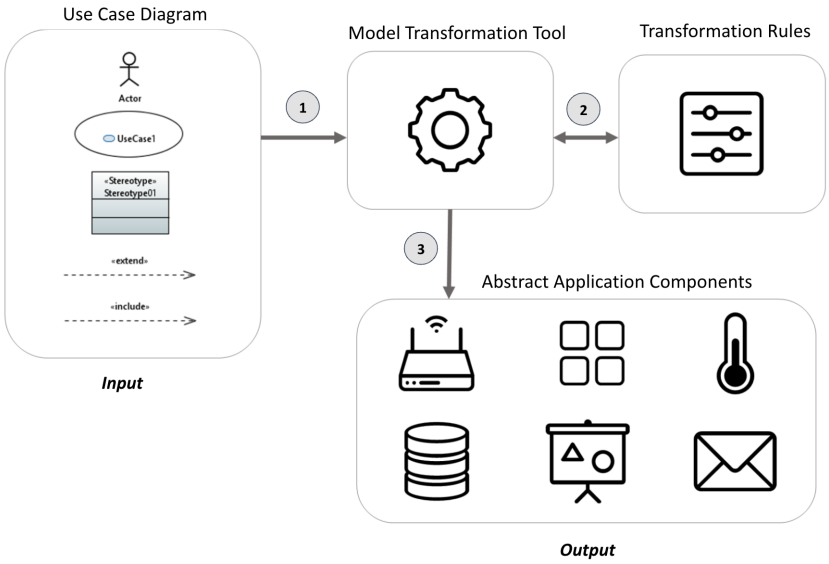


Figure 4: An overview of the model transformation process supported. The MoT's model transformation tool receives a use case diagram as input (Step 1) and uses transformation rules (Step 2) to generate abstract application components as output artifacts (Step 3).

Step 3: Component configuration. It depends on user interaction to provide additional information for a particular application component. This information is specific to a particular application, such as database connection information, sensor signing/publishing information, user information, and social network passwords. These settings are necessary because the MoT user can only represent some of the required information through the UML diagram. In this configuration, if the user lacks data storage for application data from a particular service, such as a database service, the MoT approach can provision that service with cloud computing providers. This way, users can configure cloud computing services without technical expertise. The abstraction of technical details is one of MoT's characteristics. The simplification of the CoT application development process involves abstracting cloud computing services, configurations, and components,

including software components, devices, and sensors in the context of IoT.

Step 4: Model transformation. Automatic generation of platform-specific models through a generative approach to the MDE, which aims to transform models to models or models to code. Model transformation relies on information about the application's components, and the transformation tool translates component information and configurations into platform-specific components. These models represent the same application as those involved in previous activities, but they translate higher-level abstractions into platform-specific implementations. For example, this includes conversion types for platform-specific sizes and mapping the application elements to the platform programming model. In addition to transforming the components for a specific platform, this step generates a deployment package that includes the infrastructure configuration files required to deploy and execute the application in a cloud environment.

Step 5: Application deployment. After performing the required model transformations, the MoT approach prepares the application for deployment. In practice, this step establishes the cloud computing environment for running the CoT application. This cloud environment is based on the deployment package generated in the previous step. Figure 5 presents an infrastructure model created for the cloud computing provider Amazon Web Services (AWS): API Gateway acts as a ''gateway'' for applications to access data, business logic, or functionality in their back-end; Lambda allows code execution without provisioning or managing servers; and S3 is a service for storing files. Cloud computing providers, e.g., Amazon Web Services, Google Cloud Platform, and Microsoft Azure, are heterogeneous. Developers must fully understand this heterogeneity to push CoT applications to these providers. However, this understanding is hard to acquire, making it challenging to migrate CoT applications between providers. To mitigate this problem, this step facilitates CoT application delivery by abstracting the complexity introduced by heterogeneous cloud providers and automating the creation of a runtime environment for the CoT application. If the application requires customizations before delivery, this step will also prepare it for these customizations. This delivery customization is done in the next step.

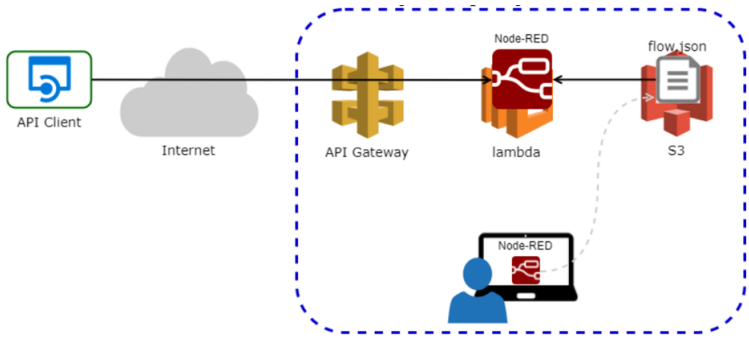


Figure 5: An architecture in which an API client communicates via API Gateway to an AWS Lambda running Node-RED, which retrieves flow configurations from S3, enabling remote workflow execution and management.

Step 6: Delivery customization. Customizations are sometimes required before delivering a CoT application, or mandatory data must be collected to enable the delivery

of CoT applications. This step adapts the application components to a specific platform. For example, sensitive user information, such as login credentials, is collected to deploy an application component to a cloud computing provider. This step implements platform-specific modifications and obtains sensitive user information that is not considered in Step 4. The next step starts if no customization or sensitive information is needed.

Step 7: Application delivery. This step ends the development cycle of a CoT application. It focuses on delivering the CoT application itself. In this sense, this step uses delivery procedures to ensure the features of CoT applications are accessible to their users. The MoT user can run the delivered CoT application locally for testing and approval. Afterward, it can be migrated to a cloud computing environment, thereby ending the development process.

4.2 UML Profile

UML [OMG 2017, Farias et al. 2017] is a general-purpose language that can model a wide range of application domains; in certain situations, it cannot model applications defined in specific domains (*e.g.*, financial applications, time applications, real multimedia applications, among others). Thus, the language needs extension. The UML can be extended in two ways. The first form is through a lightweight extension. A UML profile is created to add semantic annotations to the UML metamodel, adapting it for specific platforms (such as Spring Boot, J2EE, and EJB) or application domains (such as finance and telecommunications) by adding stereotypes, tag values, and constraints. The second one represents a heavyweight extension. The UML metamodel is modified non-conservatively, altering the standard semantics of its elements.

The MoT approach proposes a UML profile, known as *MoT.Profile*, which comprises a set of stereotypes to support the representation of CoT application concepts [Oliveira et al. 2008]. A stereotype is an extension of an existing UML metaclass or another stereotype, possibly defining additional attributes. Table 2 presents the *MoT.Profile*. In total, the MoT.Profile defines nine stereotypes organized into five categories. Each category groups stereotypes by their intended use. For example, *SensorSubscribe* and *SensorPublish* fall under the IoT category because they connect to an IoT sensor.

4.3 Key features of MoT

The key features of MoT that are provided are listed as follows:

1. **Abstraction of IoT components.** The proposed approach abstracts the complexity of the IoT components in UML models, which allows even those who do not have programming knowledge to be able, for example, to develop an application to connect to sensors and/or services offered by social networks, cloud computing, and APIs from other service providers.

2. **Automated source code generation.** Through a tool for transforming models into text, the approach generates the source code of the application based on the elaborated models. This feature enhances team productivity by enabling the reuse of models to build multiple applications and thereby ensuring higher-quality software.

3. **Setting up the environment.** Based on the components used to build the application, the approach realizes the creation and configuration of the environments in cloud computing providers to support the development of the CoT application. As such, the

Category	Stereotype	Description
IoT	≪SensorSubscribe≫	Monitors data from an IoT sensor by connecting to an MQTT Broker and signing messages from a specific topic.
IoT	≪SensorPublish≫	Sends data to an IoT sensor by connecting to an MQTT Broker and posting messages to a specific topic.
Storage	≪DatabaseSave≫	Establishes a connection to a Database server to save data.
Dashboard	≪DashboardGauge≫	Displays information in the form of a gauge on a Dashboard page.
Dashboard	≪DashboardChart≫	Displays information in the form of a chart on a Dashboard page.
Emotiv BCI	≪FacialExpression≫	With a set of Emotiv BCI tools to get brain-computer interface integration to connect a facial expression.
Emotiv BCI	≪MentalCommand≫	Using the Emotiv BCI (Brain-Computer Interface) toolkit allows to create brain-computer interface integration to capture the value of a mental command from a profile.
Social	≪TwitterPost≫	Connects to a Twitter account to post a post on the social network.
Social	≪SendEmail≫	Connects to an SMTP server to send e-mail messages.

Table 2: A list of MoT.Profile stereotypes.

approach abstracts deployment details, reducing the complexity of building native cloud applications that fully leverage cloud computing benefits such as scalability and pay-as-you-go.

4. **Extensibility.** By creating new stereotypes and corresponding transformation rules, it is possible to add new features to the approach and thereby extend the use of the approach to other domains.

The integration between UML profiles and Node-Red in MoT is not conceived as a simple engineering convenience, but as a principled architectural bridge that aligns platform-independent modeling abstractions with executable, low-code artifacts. The MoT UML profile defines a stable semantic layer for CoT concepts, enabling developers to consistently and reusably express system intent, interaction structures, and cloud behaviors. Node-RED then acts as an execution substrate in which these semantics are materialized as deployable flows. This structured mapping preserves architectural intent, reduces configuration effort, and mitigates the knowledge barrier for developers, while maintaining traceability from high-level models to executable implementations.

Although our implementation uses Node-Red as the primary execution platform, MoT was designed to remain general and reusable across heterogeneous IoT ecosystems. The stereotype-to-component mappings are declarative and repository-based, enabling extension or replacement without altering the conceptual model. This separation of concerns fosters portability and supports architectural evolution, enabling MoT to adapt to alternative cloud infrastructures and IoT development environments. By formalizing the bridge between model semantics and low-code execution, MoT moves beyond ad hoc platform use and provides a systematic, model-driven foundation for CoT application development.

4.4 Transformations

Figure 6 illustrates the mappings used to transform use-case diagrams into Node-RED components. During the MoT approach process, there are two moments where model transformations occur (Step 2 and Step 4):

1. Diagram transformation. It occurs from analyzing the elements in the use case diagram and the stereotypes used in the UML *MoT.Profile*. For each element and/or stereotype identified in this transformation, MoT performs a specific mapping of that element to an application component.

2. Models transformation. This occurs after the configuration steps of the application components. In this transformation, these components are transformed into components of the application's target platform.

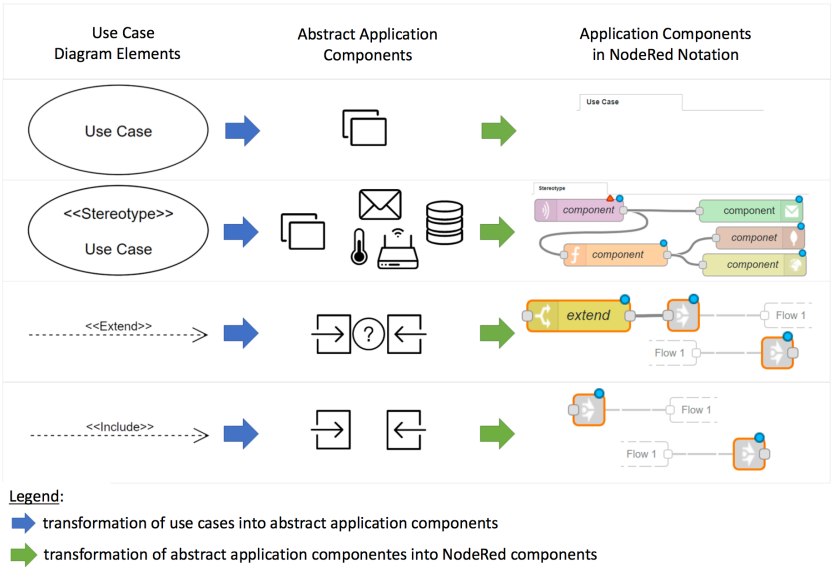


Figure 6: Illustration of the mappings performed to support the transformation of use-case diagrams into Node-RED components.

The MoT approach does not aim to establish a formal mathematical or algebraic foundation for model-driven development. Instead, it adopts a pragmatic, engineering-oriented strategy whose primary goal is to reliably transform models into executable systems through structure and automation. The tool expresses transformation behavior, execution constraints, and architectural guarantees concretely through well-defined transformation rules and controlled processing steps, rather than formal grammars or symbolic semantics. In particular, MoT operationalizes model semantics using structured JSON descriptors and a transformation pipeline that interprets UML stereotypes and transforms them into executable Node-RED flows. This design emphasizes repeatability,

ensuring that identical models consistently produce functionally equivalent deployments, while also supporting extensibility through rule-based integration of new stereotypes and components.

Transformation rules guide the generation of executable artifacts from the modeled diagrams. During this process, for each stereotype identified, MoT retrieves the corresponding JSON specification from a repository of transformation descriptors. The MoT.Transformer then interprets this file to determine whether the stereotype maps to a single component or to a composite structure formed by multiple components. For composite stereotypes, the rule evaluation proceeds recursively until the process reaches a final component that contains only executable properties. These properties capture the essential configuration parameters required later in the Component Configuration step. This mechanism not only drives the execution pipeline but also reinforces extensibility: to introduce new functionality, developers only need to define new JSON descriptors and associate them with new stereotypes in the MoT profile. Figure 7 illustrates this process for the ≪SensorSubscribe≫ stereotype.

Although MoT does not pursue numerical optimization or formal performance modeling, it operates in an environment where performance considerations are relevant. Therefore, MoT preserves architectural traceability between model constructs and execution behavior, enabling developers to reason about key characteristics of CoT applications, such as latency, payload behavior, deployment delay, and event frequency. Rather than constraining performance by design, MoT provides a structured execution basis for performance tuning, evaluation, and future performance-aware extensions. In doing so, MoT shows that model-driven abstractions can be transformed into working CoT applications with minimal manual intervention, while still enabling future research to explore optimization, formal semantics, and advanced performance strategies on this practical foundation.

4.5 Architecture

Figure 8 shows the MoT approach's architecture and its three independent modules: *MoT.Modeling*, *MoT.Transformer* and *MoT.Builder*. Each module follows well-known software design principles [Richards & Ford 2020, Martin 2018], including the single-responsibility and open-closed principles.

1. **MoT.Modeling:** This module models the CoT application. It takes as input a Use Case diagram and MoT Profile specifications. To support modeling, it integrates widely used industry tools, such as Enterprise Architect and Papyrus. The tool extends UML using UML profiles, an established extension mechanism. For example, an MoT Profile can define specific IoT components, such as sensors and actuators, within a standard UML diagram.

2. **MoT.Transformer:** This module analyzes, configures, and transforms models. First, it applies transformation rules to extract application components. Next, it configures these components with the required information, for example, by assigning a cloud storage service to data-collecting sensors. Finally, it generates a deployment package, which the next module processes. For instance, a UML model of a temperature sensor is transformed into a deployable cloud function with its corresponding API endpoint.

3. **MoT.Builder:** This module creates execution environments, customizes components, and publishes the application. It automates deployment using tools such as Docker

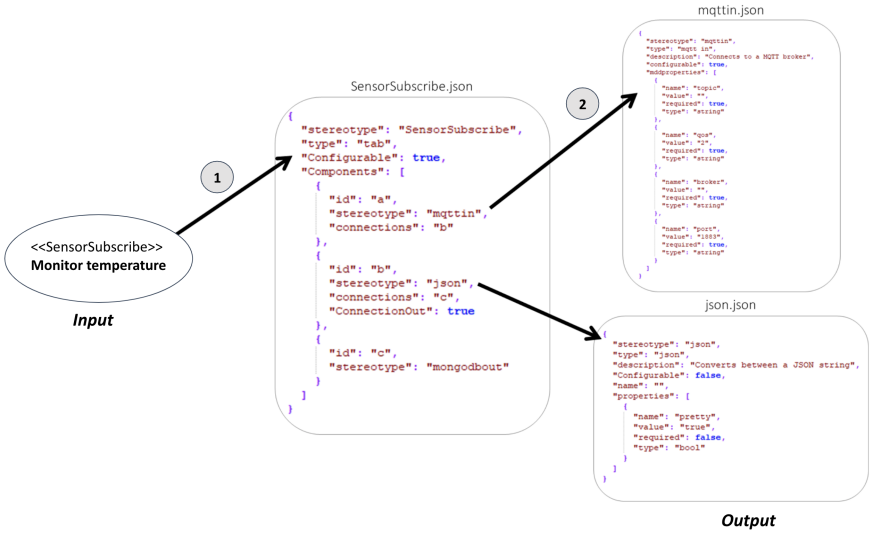


Figure 7: Transforming the use case Monitor temperature containing ≪SensorSubscribe≫ stereotype into components. The MoT receives a use case, labeled with a stereotype, as input, and accesses the ≪SensorSubscribe≫ stereotype mapping (Step 1). Next, it applies transformation rules to generate components as output artifacts (Step 2).

and Kubernetes, starting from the generated deployment package. For example, after transforming the model, MoT.Builder can create a containerized application and deploy it to a cloud service such as AWS or Azure.

To clarify MoT's architecture, we explicitly describe how the approach structures its transformation, execution, and deployment pipeline. MoT separates concerns into three well-defined layers (Figure 8). The **modeling layer** captures IoT and cloud concepts using UML stereotypes, preserving semantic intent in a technology-agnostic manner. The **transformation layer** codifies these semantics into executable artifacts through organized, rule-based mappings, associating each stereotype with reusable components and well-defined responsibilities. This modular design clarifies execution behavior, improves maintainability, and supports consistent deployment across devices. The current literature recommends modularizing modern applications [Urdangarin et al. 2021]. Finally, the **deployment layer** automates generating and configuring target artifacts, ensuring consistency during execution and reducing human intervention. Although the current implementation targets a specific runtime platform, the layered organization, rule-based mappings, and stereotype-driven semantics provide a clear path toward extensibility across heterogeneous cloud environments.

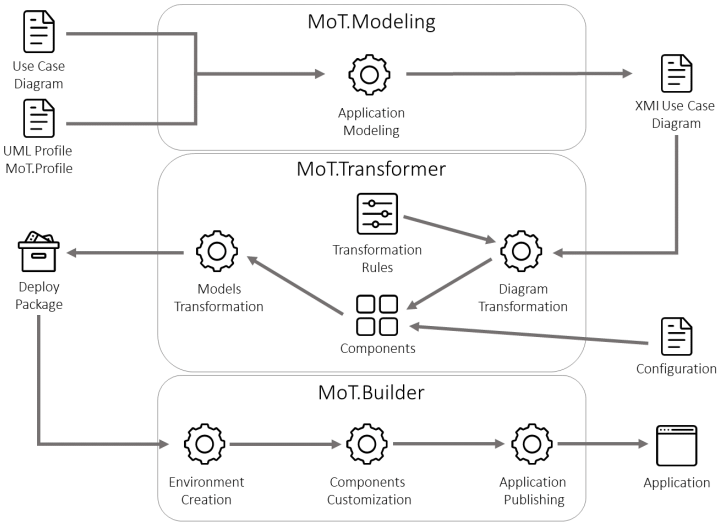


Figure 8: Overview of the MoT architecture, illustrating the three core modules — MoT.Modeling, MoT.Transformer, and MoT.Builder — along with their interactions in modeling, transforming, and deploying CoT applications.

4.6 Implementation aspects

This section outlines the technologies that were used to implement the proposed approach. We seek to explain how MoT Builder[4], MoT Transformer[5], and MoT Profile[6] were implemented. We implemented the MoT approach in C# using ASP.NET and Visual Studio. The following section discusses the technologies used to implement each MoT module.

MoT.Profile. The UML profile of the approach was crafted using Eclipse Papyrus[7], an Eclipse-based free software UML modeling environment that supports creating UML profiles.

MoT.Modeling. This module aims to generate the models of the application to be created. This modeling can be performed with tools widely used in the software industry, such as Eclipse Papyrus, Astah, and Modelio, among others available on the market. The tool supports the UML language and extends it via UML profiles, which extend the traditional UML standard. It also exports templates in accordance with the XMI standard, an OMG standard for XML-based information exchange. The main responsibilities of this tool are:

1. Modeling the application. Through the use of UML and the UML profile of the proposed approach, the requirements are formalized in models that will be represented in a diagram.

[4] MoT Builder: https://github.com/cristianowelter/MoT.Builder
[5] MoT Transformer: https://github.com/cristianowelter/MoT.Transformer
[6] MoT Profile: https://github.com/cristianowelter/MoT.Profile
[7] Papyrus: https://www.eclipse.org/papyrus

2. Exportation of the models. From the created models, the information in the diagrams is exported to XMI files, which the transformation module uses.

MoT.Transformer. It provides functionality for managing applications created by the approach. For this, a tool was implemented that has the responsibility of interpreting the XMI files generated by the UML models in the modeling step, as well as performing the validation and configuration of the components, and at the end of the configuration process, carrying out the transformation of this information to a file that will be used by the next tool. This tool consists of a Web application that separates content (HTML), presentation (CSS 3), and functions (JavaScript). The most popular front-end component library, Bootstrap, has been used to create responsive Web projects. Bootstrap provides user-friendly, responsive interfaces to web applications. The approach used plugins created with jQuery[8], an open-source JavaScript library designed to simplify client-side scripting. The tool was implemented using Visual Studio Community, the C# programming language, and ASP.NET Core, a free, cross-platform, high-performance software framework for creating modern cloud applications. To store the information, the tool uses a Microsoft SQL Server database. The main responsibilities of this tool are:

1. **Interpretation of the UML model.** The tool can interpret the XMI file exported by the UML modeling tool and, using the rules defined by the approach's UML profile, generate the pre-defined application components to be created.
2. **Components configuration.** Based on the components identified in the interpretation of the UML model, the tool asks the user for the necessary, specific information for each component. This information will complement the UML model. This configuration distinguishes one application from another because the same UML model can be used to create multiple applications, but these settings make the application unique.
3. **Models transformation.** After the configuration of all the components of the application to be generated, the tool performs the export of the information collected to a deployment package, which contains the information needed to build the application through the *MoT.Builder* module.

Figure 9 shows the application management interface of the implemented tool. It is possible to identify the options for managing a CoT application; these options are: *Details*: displays the details of the application; *Models*: option to import the models created from the modeling step; *Components*: option to access the application's component list and configure them; and *Cloud Providers*: option to configure the data of a cloud computing service provider to create the infrastructure for execution.

MoT.Builder. The last module comprises executing the application, customizing the components, and ending its publication. This step requires two tools, both running on the Node.js platform, enabling JavaScript applications to run on the server. The first tool is Node-Red, a programming tool for connecting hardware devices, APIs, and online services, with a browser-based editor for connecting components and generating a stream. The Node-RED interface displays the application flow generated in the previous step, allowing the user to run the application, identify implementation issues, and, if necessary, customize the generated components. The second tool to be used in this step, and which also runs on the same platform, is the Serverless Framework[9], a toolkit for deploying

[8] https://jquery.com/
[9] https://www.serverless.com/

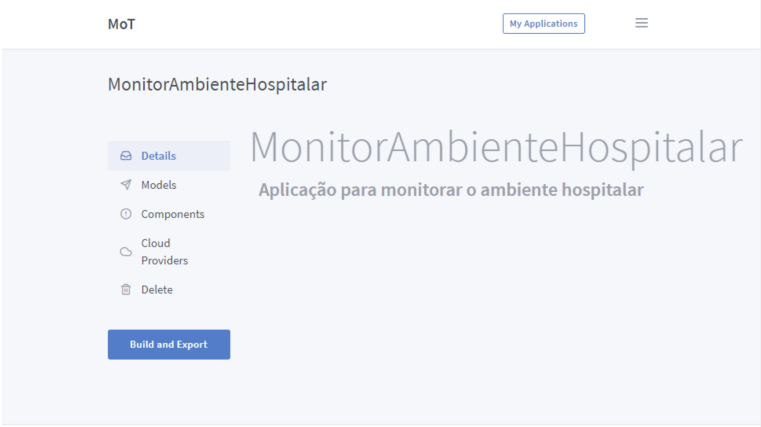


Figure 9: Interface for application management of the MoT.Transformer tool.

serverless architectures across any cloud hosting provider that automates infrastructure configuration. The main responsibilities of these tools are:

1. Execution of the application. Through the Node-RED platform, you can run the generated application even outside a production environment; this helps identify flow issues, configuration problems, and application architecture failures.
2. Customization of components. If required, you can edit the application components and their settings via the Node-RED interface. We can also extend the application's functionality by adding new components, functions, and services. Still, these changes require some technical knowledge of JavaScript and familiarity with the platform.
3. Application deployment. After verifying and validating the application's functionalities, it can be deployed through the Serverless Framework tools. This process is automated and provides a link to access the generated application.

4.7 Limitations

The current version of MoT adopts a deliberately engineering-oriented design, prioritizing feasibility and demonstrable automation over full formal expressiveness and adaptive transformation intelligence. Transformation logic is implemented via predefined mappings, which validate the approach's technical viability but constrain flexibility. Currently, MoT lacks dynamic adaptation or configurable transformation strategies that automatically adjust to alternative architectural choices, unforeseen modeling patterns, or platform-specific variations. Likewise, although we clarify the transformation process's internal behavior, the approach does not yet formalize its semantics or provide mathematically grounded validation mechanisms. These constraints suggest that MoT is a feasible, low-code proof of concept rather than a fully configurable, formally verified transformation framework.

Furthermore, the current implementation offers limited fault tolerance and operational resilience. MoT does not incorporate automated rollback procedures, robustness

mechanisms to handle partial failures, or recovery workflows for incomplete model configurations or incorrect deployments. Error handling relies primarily on user intervention rather than autonomous platform support. In addition, this evaluation does not include quantitative performance indicators such as transformation latency, scalability under load, or execution failure rates. These omissions reflect the study's focus on feasibility and usability rather than exhaustive performance benchmarking. We explicitly acknowledge these restrictions to make the operational boundaries of MoT transparent and to guide readers regarding its current maturity and intended use.

Problem formulation and evaluation scope limitations. A further limitation of this study concerns the scope and nature of its problem formulation and evaluation. The present work adopts a descriptive, engineering-oriented perspective, prioritizing feasibility and demonstrability over a formal optimization framing. Accordingly, we do not explicitly define measurable optimization objectives or constraints, such as reducing deployment latency, increasing automation coverage, reducing cloud costs, or improving bandwidth efficiency. Likewise, although the mapping between UML requirements and executable artifacts is systematic and rule-based, it has not yet been formally evaluated for correctness, coverage, or completeness. The study also does not quantify reductions in developer effort or provide baseline comparisons against alternative approaches. These omissions limit the ability to substantiate claims regarding efficiency gains and automation benefits with empirical metrics. We recognize these constraints and explicitly acknowledge that a more formal problem definition, benchmark-driven validation, and quantitative assessment remain essential directions for future work.

Evaluation scope and external validity limitations. Although the proposed approach demonstrates feasibility, we acknowledge essential limitations regarding the breadth and depth of its evaluation. The current study does not measure scalability, performance behavior under different workloads, or resilience in multi-configuration stress scenarios. Its empirical assessment relies on a single application domain and a constrained experimental setting, limiting external validity and reducing our ability to generalize the findings. We also recognize challenges in recruiting participants with expertise in both modeling and CoT technologies, the dependence on Node-Red as a demonstration platform, and the absence of strong comparable MDE tools for fair benchmarking. For these reasons, this work should be understood primarily as a feasibility and proof-of-concept investigation rather than a comprehensive validation study. Nonetheless, it establishes a necessary foundation by showing that the approach is technically executable end-to-end and meaningful to knowledgeable practitioners. Future work will address these limitations through multi-scenario case studies, quantitative benchmarks, comparative experimentation with alternative MDE solutions, and evaluation across heterogeneous IoT/cloud environments.

5 Evaluation

This section presents the key decisions underlying the experimental procedures used to evaluate the proposed approach. We evaluated our approach through a case study developing a real-world CoT application (Section 5.1). In addition, we assess the MoT approach with respect to users' perceptions of usability and acceptability (Section 5.2). Our evaluation methodology adhered to practices established in previously published articles and followed widely known guidelines [Wohlin et al. 2012, D'Avila et al. 2020, Farias et al. 2015, Rubert and Farias 2022].

5.1 Case study for Monitoring of Hospital Environments

Scenario. Continuous monitoring is essential in hospital environments to collect biometrics [Menzen et al. 2021], as even minor failures can jeopardize lives. Regulations specify the environmental conditions required for patient care, vaccine storage, and medication preservation, which require precise temperature and humidity control. However, many hospital units still rely on manual monitoring. Automating this monitoring process presents a robust solution to these challenges. Key features include automated analysis, graph-based visualization, and customizable alert mechanisms. The system's operational framework involves collecting, storing, and processing IoT sensor data for continuous monitoring and informed responses. From this scenario, we derive the following functional requirements for the proposed application:

- Monitor environmental temperature.
- Store temperature data.
- Generate graphical representations of temperature data.
- Issue alerts for critical temperature variations.

We developed a CoT application using the proposed approach to fulfill the requirements of the hospital monitoring scenario. We detail the step-by-step process as follows:

5.1.1 Step 1: Application modeling

Figure 10 illustrates a UML use case diagram for a temperature monitoring system with core functionalities and their relationships. We used the open-source tool Eclipse Papyrus to elaborate on it. The primary actor, labeled as *User*, interacts with the central use case, *Temperature Monitoring*, which encompasses three related functionalities:

- **Save Data**: The *Temperature Monitoring* use case includes this functionality, represented by an `<<include>>` relationship. This indicates that data saving is a mandatory sub-process whenever temperature monitoring occurs. It ensures that all monitored temperature data is recorded persistently.
- **Show Chart**: Another `<<include>>` relationship links the *Show Chart* functionality to the *Temperature Monitoring* use case, indicating that generating and displaying charts is an essential part of the monitoring process. This function provides users with a visual representation of temperature data over time, enhancing the system's usability.
- **Send Notification**: The *Send Notification* functionality is associated with *Temperature Monitoring* through an `<<extend>>` relationship. This relationship implies that notifications are sent only under specific conditions, such as exceeding temperature thresholds. It allows the system to provide timely alerts without requiring every monitoring instance to trigger a notification.

Importing the *MoT.Profile* makes it possible to enhance the use-case diagram with semantic annotations provided by the profile. These annotations provide high-level abstractions that describe the application's functionality within a CoT environment without detailing implementation details. Figure 11 presents the annotated use-case diagram,

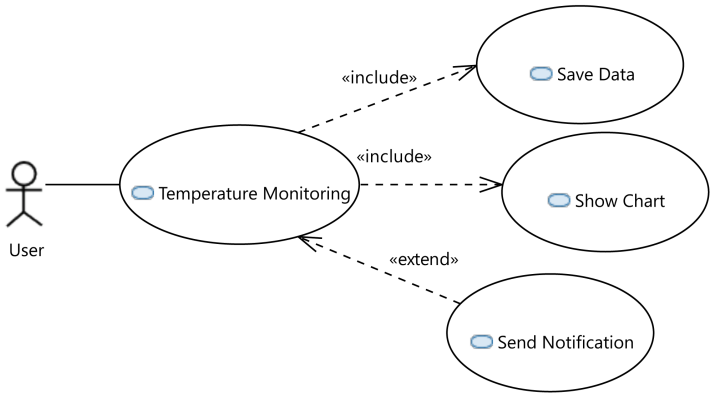


Figure 10: Illustration of a use case diagram for temperature monitoring. The user monitors temperature, including data storage and graph visualization. Notifications extend monitoring, triggering alerts when predefined conditions are met.

showing the use of the following stereotypes: *SensorSubscribe*, *DatabaseSave*, *DashboardGauge*, and *SendEmail*. These annotations will inform the model transformation processes performed in subsequent steps. Once the use-case diagram is complete, it is exported as an XMI document directly from the UML modeling tool. This file captures all diagram elements in a format that a transformer can interpret. Figure 12 shows the exported file, where the diagram elements and applied stereotypes are represented as text.

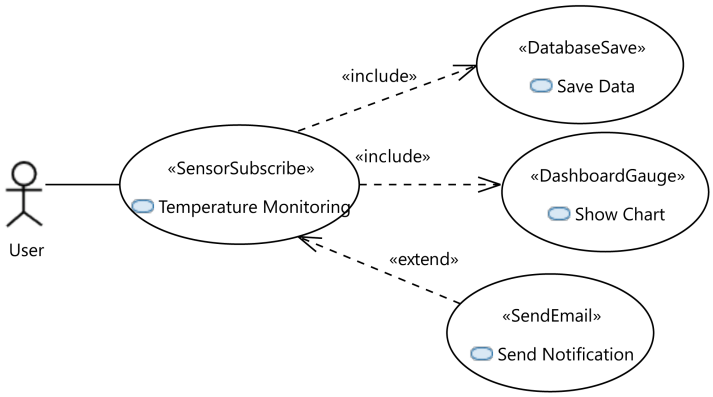


Figure 11: Detailing the temperature monitoring use case using the proposed MoT.Profile stereotypes.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="20131001" xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <uml:Model xmi:id="_ERHn0BwjEemrk7v8KLXFVg" name="MonitorAmbienteHospitalar">
    <packageImport xmi:type="uml:PackageImport" xmi:id="_Ee7aUBwjEemrk7v8KLXFVg">
      <importedPackage xmi:type="uml:Model" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
    </packageImport>
    <packagedElement xmi:type="uml:Actor" xmi:id="_GSitABwjEemrk7v8KLXFVg" name="User"/>
    <packagedElement xmi:type="uml:UseCase" xmi:id="_UyZo0BwjEemrk7v8KLXFVg" name="Temperature Monitoring">
      <extensionPoint xmi:type="uml:ExtensionPoint" xmi:id="_ob9pYBwjEemrk7v8KLXFVg" name="ExtensionPoint2"/>
      <include xmi:type="uml:Include" xmi:id="_ngwlYBwjEemrk7v8KLXFVg" addition="_hYlGMBwjEemrk7v8KLXFVg"/>
      <include xmi:type="uml:Include" xmi:id="_d5eVUDlfEem-dNGzL9zigg" addition="_XW-4sDlfEem-dNGzL9zigg"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Association" xmi:id="_eQLx4BwjEemrk7v8KLXFVg" memberEnd="_eQTGoBwjEemrk7v8KLXFVg
    _eQTGoRwjEemrk7v8KLXFVg">
      <ownedEnd xmi:type="uml:Property" xmi:id="_eQTGoBwjEemrk7v8KLXFVg" name="Temperature Monitoring"
        type="_UyZo0BwjEemrk7v8KLXFVg" association="_eQLx4BwjEemrk7v8KLXFVg"/>
      <ownedEnd xmi:type="uml:Property" xmi:id="_eQTGoRwjEemrk7v8KLXFVg" name="User"
        type="_GSitABwjEemrk7v8KLXFVg" association="_eQLx4BwjEemrk7v8KLXFVg"/>
    </packagedElement>
    <packagedElement xmi:type="uml:UseCase" xmi:id="_hYlGMBwjEemrk7v8KLXFVg" name="Show Chart"/>
    <packagedElement xmi:type="uml:UseCase" xmi:id="_kAlqQBwjEemrk7v8KLXFVg" name="Send Notification">
      <extend xmi:type="uml:Extend" xmi:id="_ob8bQBwjEemrk7v8KLXFVg" extendedCase="_UyZo0BwjEemrk7v8KLXFVg"
      extensionLocation="_ob9pYBwjEemrk7v8KLXFVg"/>
    </packagedElement>
    <packagedElement xmi:type="uml:UseCase" xmi:id="_XW-4sDlfEem-dNGzL9zigg" name="Save Data"/>
  </uml:Model>
  <MoT:SensorSubscribe xmi:id="_wJ4kIEnSEemRzMbvzg8ROA" base_UseCase="_UyZo0BwjEemrk7v8KLXFVg"/>
  <MoT:DatabaseSave xmi:id="_xUsKYEnSEemRzMbvzg8ROA" base_UseCase="_XW-4sDlfEem-dNGzL9zigg"/>
  <MoT:DashboardGauge xmi:id="_yj5IcEnSEemRzMbvzg8ROA" base_UseCase="_hYlGMBwjEemrk7v8KLXFVg"/>
  <MoT:SendEmail xmi:id="_zff1EEnSEemRzMbvzg8ROA" base_UseCase="_kAlqQBwjEemrk7v8KLXFVg"/>
</xmi:XMI>
```

Figure 12: XMI representing the elements of the use case diagram.

5.1.2 Step 2: Diagram transformation

We use the *MoT.Transformer* module to generate the application. By importing the XMI file from the modeling step, the proposed tool performs the first model transformation defined by this approach. In this process, the transformer maps all diagram elements and converts those with UML profile-compatible stereotypes into application components. Figure 13 shows a prototype screen displaying these components, which are transformed from the input XMI file. Figure 13 highlights the diagram's use cases and components created to fulfill the functionality specified by the applied stereotypes.

5.1.3 Step 3: Component configuration

After transforming the diagram, the target application includes components required to build each feature defined in the application model. These components vary with the stereotype used and may include functions, cloud services, connectors to other services, and additional components. Not all components are ready for use; some require additional user input. *MoT.Transformer* provides features to help fill in this information. As shown in Figure 13, components that require configuration feature a button that directs the user to a form requesting the required details. Figure 14 illustrates the information needed for the e-mail component, including the *Send Notification* use case.

As shown in Figure 14, one key benefit of the MoT approach is abstracting the complexity of specifying programming languages and deployment platforms. With this approach, configuring a resource such as email sending reduces to filling out a simple form, without requiring users to handle technical details. Sensitive information, such as the username and password, has been removed from this stage and must be provided in subsequent steps. Beyond component configuration, another strength of the MoT approach is its ability to abstract the diversity of service configurations across cloud computing providers. By requesting specific information in the form, users can instantiate the services the application needs.

Figure 15 shows the configuration of the MongoDB output component, which represents a database instance—a service commonly provided by cloud providers. When

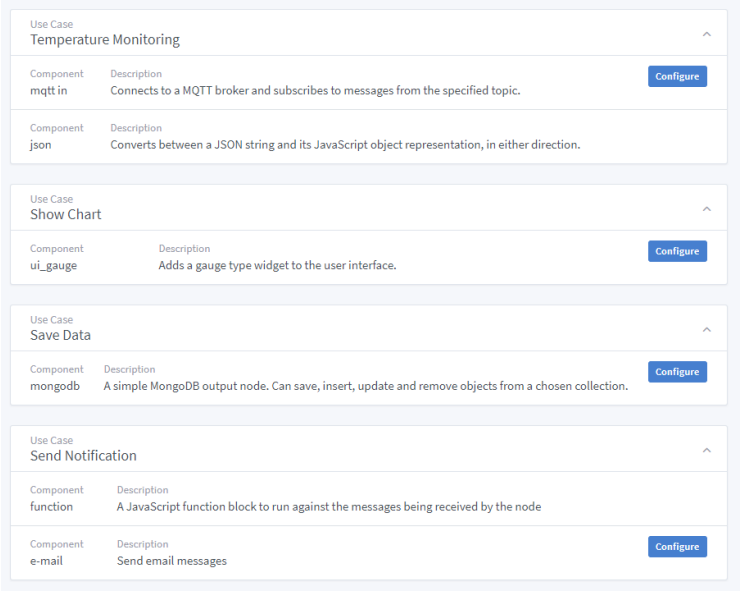


Figure 13: List of application components generated after the diagram transformation process, detailing the identified elements and their configurations for deployment.

configuring this component, users can add a new server. Selecting this option brings up a new form, shown in Figure 16, where users can choose a cloud service provider to create a new service instance. Once a provider is selected, the *MoT.Transformer* module will configure the service and integrate it with the application component using the connection details.

Step 4: Model transformation. The MoT approach transforms models through the *MoT.Transformer* module. At this stage, it converts the application's components and their configurations into platform-specific components. The approach supports generating applications for the Node-Red platform, including the resources required to run them. In addition to transforming platform-specific components, this step generates the implementation package, which contains infrastructure configuration files for deploying and running the application in a cloud environment. This package includes several tools and frameworks that run on the Node.js platform and are part of the *MoT.Builder* module. It provides the application's file structure, including configuration files, scripts, package dependencies, and more.

Step 5: Application deployment. The *MoT.Builder* module handles the application's deployment. It includes the Node.js JavaScript runtime and various dependency packages, such as Node-Red and the Serverless Framework, to build the structure. All configuration files and dependencies are included in the implementation package, generated during the model transformation stage. To run the application, run two commands in the operating system's terminal. The first command is *npm run setup*, which installs all necessary dependencies and configures the cloud computing model environment for application execution. The second command is node *./node_modules/nodered/red.js -s ./settings.js*. This prepares a local environment to execute the configurations. The Node-Red platform will be accessible at *http://localhost:1880*. This environment allows customization of

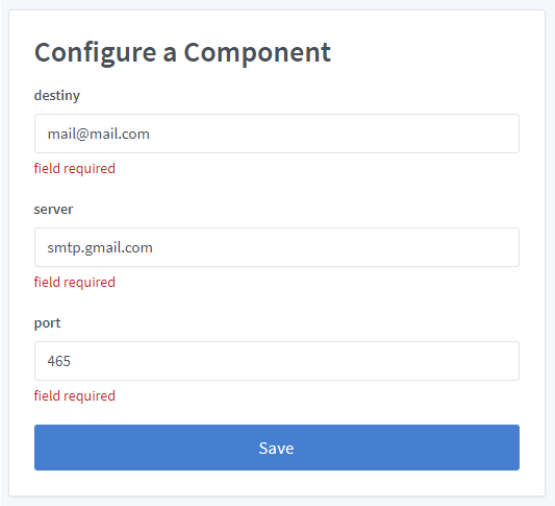


Figure 14: User interface for configuring the email component, specifying parameters such as sender details, SMTP settings, and port.

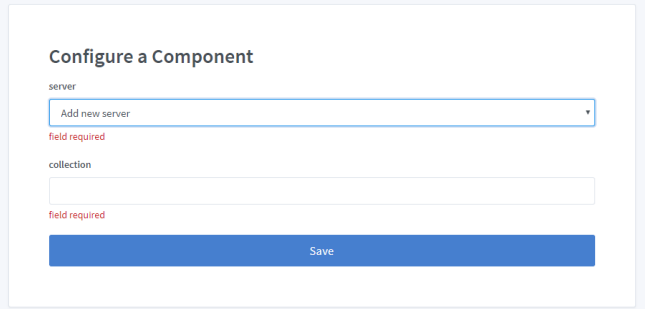


Figure 15: Interface for configuring the MongoDB component, specifying server details and target collection for data storage.

components in the Node-Red application.

Step 6: Component customization. Component customization is done through the Node-Red interface, accessible via the localhost address. Figure 17 shows the Node-Red interface, where each use case from the diagram is mapped to a specific tab, and the component flow represents the functionality of the stereotype assigned to the use case. The figure for the Temperature Monitor use case shows the configured elements and their connections with other flows. Figure 18 details each use case and the components corresponding to each functionality.

Component customization is performed through the Node-Red interface, accessible at localhost. Figure 17 shows the Node-Red interface, where each use case in the diagram is mapped to a specific tab, and the component flow represents the functionality of the stereotype attributed to the use case. The figure shows the Temperature Monitor use case, the configured elements, and the elements representing connections to the other

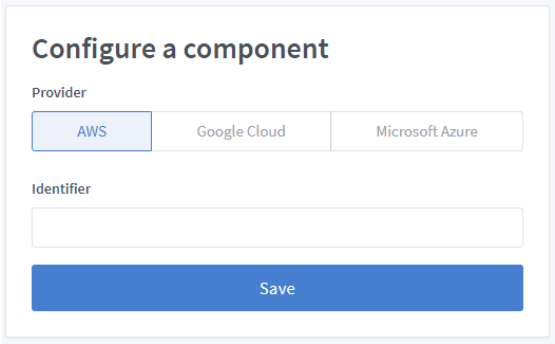


Figure 16: Configuring a new MongoDB service instance by selecting the cloud provider and defining a unique identifier.

fluxes. Figure 18 shows the details corresponding to each use case and the components corresponding to each functionality.

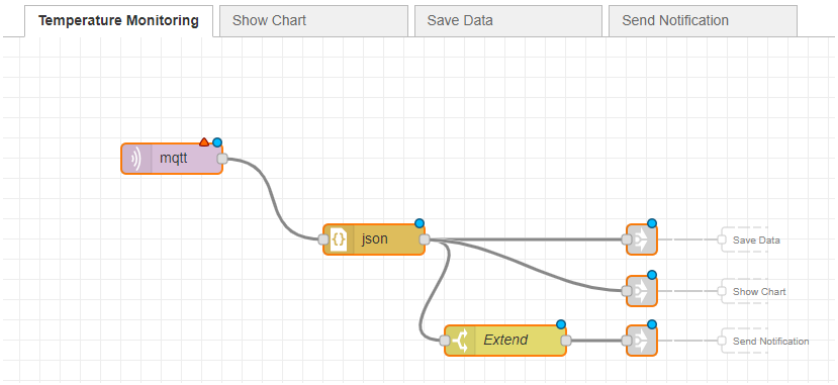
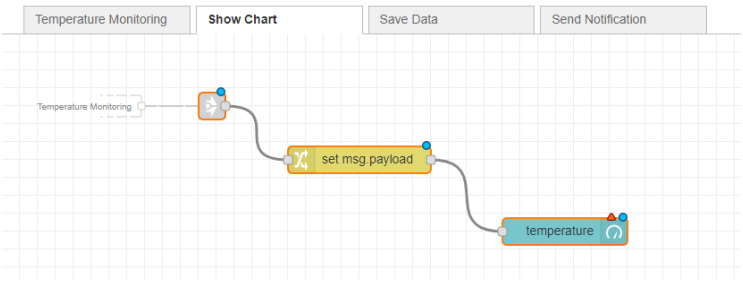


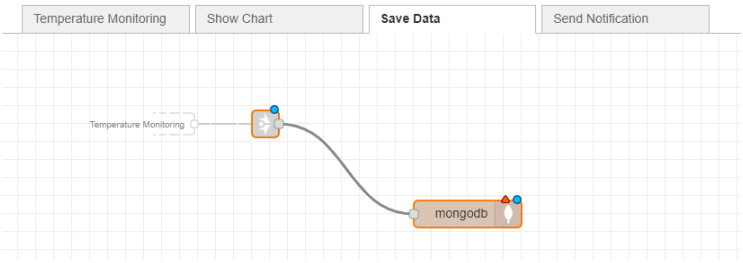
Figure 17: Visual representation of a temperature monitoring workflow implemented in the Node-Red platform.

As noted in the component configuration step, some sensitive information, such as login credentials, was not configured earlier. Adjustments must be made at this stage for components requiring this data. Figure 19(a) shows the standard Node-RED form, where users can fill in the previously missing fields. Figure 19(b) displays the values used to configure the method extension clause for sending a notification when the temperature changes. At this stage, in addition to customizing the components generated by the approach, Node-Red provides other components that can be added to the application flow to create new functionalities not foreseen by the approach. Once the component customization is complete, the application is ready for execution. This marks the final step of the approach: executing the application.

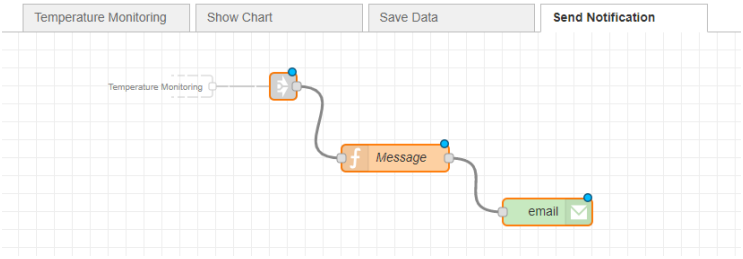
Step 7: Application execution. The final step involves executing the application generated through the established process. Execution begins by pressing the *Deploy*



(a) Show Chart



(b) Save Data



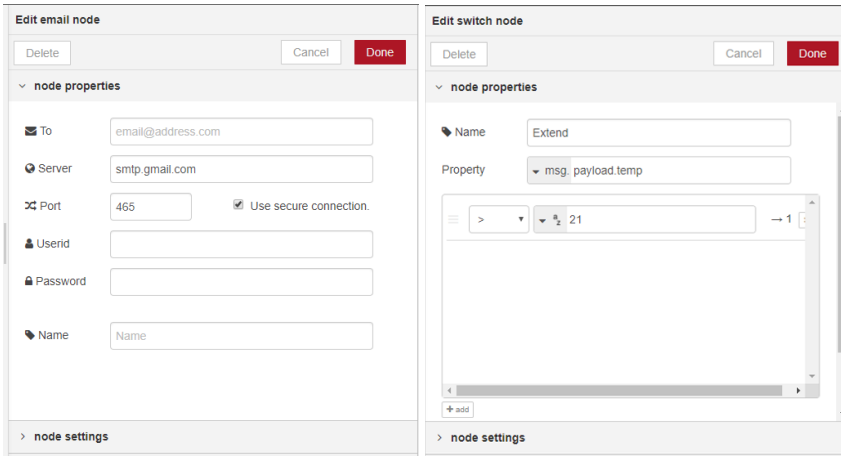
(c) Send Notification

Figure 18: Flows automatically generated in Node-RED for key functionalities: (a) Show Chart, (b) Save Data, and (c) Send Notification.

button in Node-Red, which starts the local application and updates the environment files set up during the implementation stage. Figure 20 and Figure 21 show the final result after deploying the application in Node-Red. They display an ambient temperature indicator based on the sensor data. In addition to the indicator, when the temperature changes according to the flow extension rule, an email is sent with the details provided in the configuration step.

5.2 Acceptance of the proposed approach

We evaluated the MoT approach in an empirical study of 12 participants to assess its acceptance. This evaluation used the Technology Acceptance Model (TAM) [Marangunić & Granić 2015] as the framework. The article employed methodologies previously



(a) SMTP Config (b) Extend Use Case

Figure 19: Customization options for the e-mail component: (a) SMTP Configuration and (b) Use Case Extension.

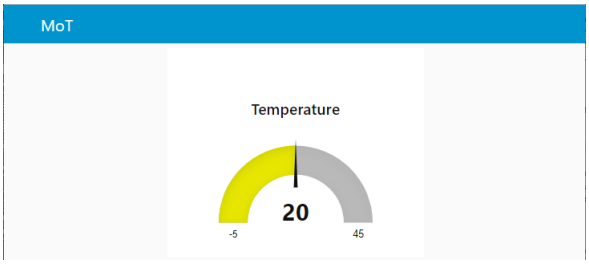


Figure 20: Visualization of recorded temperature data over time.

validated in published research to assess technology acceptance, utilizing the TAM questionnaire as a standardized evaluation tool [D'Avila et al. 2020, Vieira & Farias 2025, Farias et al. 2019]. Through the TAM questionnaire, we measured perceived ease of use, perceived usefulness, attitude toward use, and behavioral intention to use the MoT approach. Figure 22 illustrates the experimental process. The phases of the study are outlined as follows:

1. **Phase 1: Training.** This phase focuses on explaining the experimental study and training participants with the MoT approach. Participants must know their role in the experiment and what is expected of them. At this stage, the authors carefully explained the MoT approach in detail and how to use the *MoT.Transformer* and *MoT.Builder* tools.

2. **Phase 2: Apply MoT approach.** Each participant applied the MoT approach to implement the assessment scenario presented in Section 5.2.2. Participants received the application's functional requirements (input data) and were guided through the MoT development process, including all steps required to implement the application.

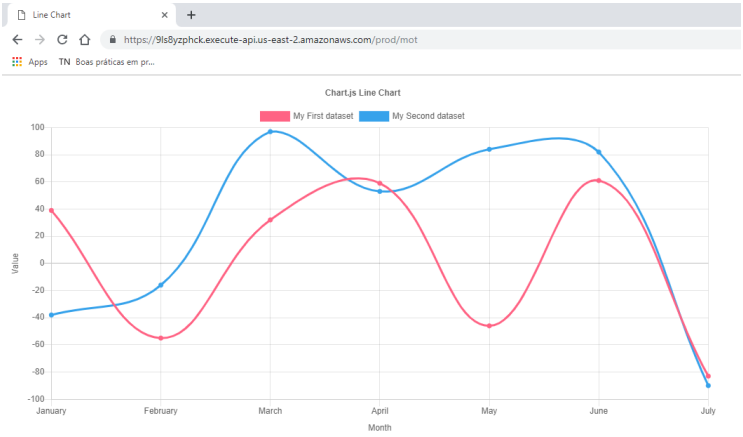


Figure 21: Visualization of a static chart application running on the Node-Red platform in a cloud environment.

At the end of this phase, participants are expected to have a complete understanding of the approach and, in doing so, can assess MoT in the next phase. We understand that participants need to use the approach to assess its usefulness.

3. **Phase 3: Apply questionnaire.** This phase has two activities. The first activity collects participants' demographic data. Participants answered a set of questions (input) to characterize them. The collected demographic data (output) is the result of this activity. The second activity uses the technology acceptance model (TAM) as input data. Participants receive a list of questions about perceptions of usability and utility, as well as behavioral intentions related to the prototype. Qualitative data (data output) are generated from users' perspectives on the prototype's usability and acceptance.

5.2.1 Questionnaires

We administered two questionnaires: one to characterize participants and another to assess the MoT approach using the Technology Acceptance Model (TAM).

Questionnaire 1: Participant profile. Table 3 presents the questionnaire used to collect data regarding the participant profile. Analyzing this data is crucial for selecting appropriate candidates for MoT testing. The questions cover general demographics, including age, gender, experience, and educational background, as well as familiarity with IoT technologies, cloud computing, and relevant professional experience. Constructing accurate user profiles ensures that the technique is evaluated by individuals whose backgrounds align with those of potential MoT users.

Questionnaire 2: Technology acceptance model. Table 4 presents the questionnaire used to assess the usability and acceptability of the approach, based on the Technology Acceptance Model (TAM) proposed by Davis [Davis 1989] and reviewed by Marangunic and Granic [Marangunić & Granić 2015]. The TAM identifies two key factors influencing technology acceptance: (a) perceived ease of use, which reflects the belief that the technology reduces effort, and (b) perceived usefulness, which refers to the belief that the technology enhances performance in task execution.

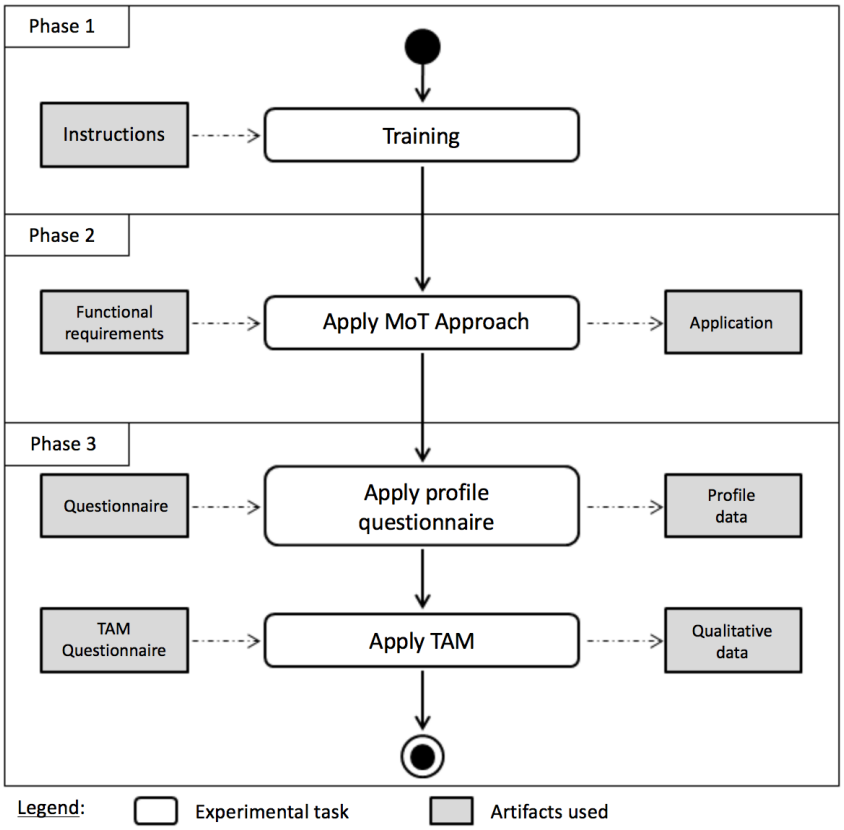


Figure 22: Illustration of the experimental process adopted to evaluate the proposed approach, detailing key phases from setup to analysis.

5.2.2 Evaluation Scenarios

The study participants applied the proposed approach across four evaluation scenarios, allowing them to use and access all its functionalities before completing the TAM evaluation questionnaire.

- **Scenario 01: Application modeling.** This scenario involves extending the traditional use case diagram by incorporating stereotypes from the MoT UML profile using a standard UML modeling tool. The user adds the appropriate stereotypes to each feature in the use case diagram.
- **Scenario 02: Diagram transformation and component configuration.** In this scenario, the user utilizes the *MoT.Transformer* prototype to import the XMI file from the use case diagram and configure the application components. The user imports the XMI representation and completes the component configurations via the respective configuration forms.

– **Scenario 03: Model transformation and application deployment.** Users employ the *MoT.Builder* tool to implement the application and construct its CoT. The user exports the deployment package into the tool and executes commands to generate scripts that create the environments required for the application's execution.

– **Scenario 04: Customization of components.** This scenario presents greater complexity because it involves specific technical knowledge of the platform to customize the application's components. The user is prompted to customize the Extend component to assign a value to direct the application flow to the new flow.

Category	Number	Question
Characterization	1	What is your age?
	2	What is your profession?
	3	What is your education level?
Experience	4	What is your experience with systems analysis?
	5	What is your experience with software development?
	6	What is your experience with the Internet of Things (IoT)?
	7	What is your experience with Cloud Computing?

Table 3: Questionnaire used to characterize participant profiles and experience levels.

Category	Number	Question
System Usefulness	8	The MoT approach would facilitate the development of CoT applications.
	9	The MoT approach would help improve developer productivity.
	10	The MoT approach would reduce the time required to deliver a CoT application.
	11	The abstraction of IoT components reduces development complexity.
	12	The abstraction of Cloud Computing components reduces the complexity of configuring environments.
Ease of Use	13	The MoT prototype is easy to use.
	14	The MoT approach is easy to learn.
	15	The MoT approach is easy to master.
Overall Perception	16	The MoT approach could be used for CoT application development in the software industry.
	17	A person with little technical knowledge in IoT and Cloud Computing could build a domestic CoT application using MoT.
	18	I would use the MoT approach to build my CoT application.

Table 4: Questionnaire for assessing perceived usefulness, ease of use, and overall perception of MoT.

5.2.3 Selection of Participants

Participants were selected using convenience sampling from university students with a background in information technology and technology professionals. All participants had experience in software development. Thus, participants with diverse knowledge

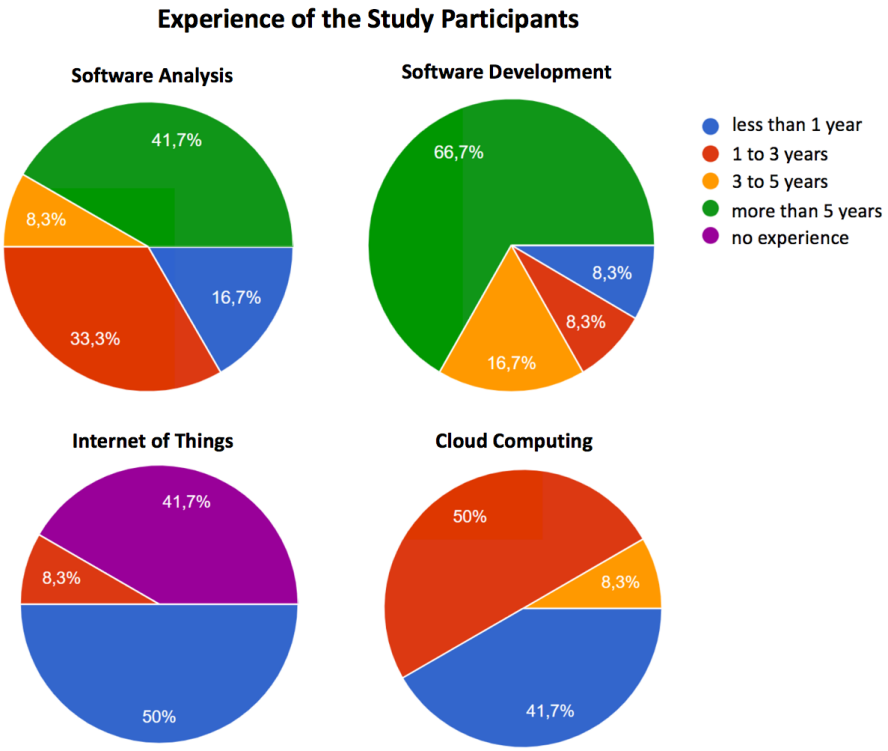


Figure 23: Overview of the study participants' experience on a year scale in software analysis, software development, internet of things, and cloud computing.

and experience profiles could be obtained. Figure 23 presents an overview of the study participants.

A total of 12 participants took part in the MoT evaluation process. Of these, 10 (83.3%) were male, and 2 (16.7%) were female, providing a diverse gender representation. Participants' ages ranged from 25 to 40 years, reflecting a balanced mix of experience levels. All participants held at least a bachelor's degree in Technology-related fields, ensuring a strong academic foundation. Professionally, 11 participants (91.7%) worked at technology companies across various sectors, while 1 participant (8.3%) was a student, providing a fresh perspective for the evaluation.

The age distribution of the participants was as follows: 6 individuals were aged 25 to 29, 2 were aged 30 to 34, 3 were aged 35 to 39, and 1 was aged 40 to 44. Regarding educational background, the majority—9 participants—held an undergraduate degree in Technology, while 3 had a Master's degree. Professionally, 11 of the 12 participants were in the technology sector, including 6 programmers/systems developers, 4 systems analysts, and 1 software engineer. The remaining participant was a student, adding valuable diversity to the group.

Regarding professional experience in systems analysis, 5 participants had over 5 years, one had between 3 and 5 years, three had between 1 and 3 years, and 2 had less than 1 year. Regarding software development, 8 participants had more than 5 years of

experience, 2 had between 3 and 5 years, 1 had between 1 and 3 years, and 1 had less than 1 year. Regarding IoT expertise, 5 participants had no experience, 6 had less than 1 year of experience, and 1 had between 1 and 3 years of experience. In cloud computing, the experience levels were as follows: five participants had less than 1 year of experience, six had between 1 and 3 years of experience, and one had between 3 and 5 years of experience. This diversity in experience levels across different areas highlights a broad spectrum of expertise within the group.

5.2.4 Results

This section presents results on participants' perceptions of ease of use, utility, and behavioral intentions regarding the MoT approach. Table 5 summarizes the data collected on participants' perceived ease of use. Notably, 100% of interviewees agreed that the prototype is easy to use and requires no significant effort to learn its features. Specifically, 50% of participants partially agreed, while the remaining 50% fully agreed. Furthermore, when considering the potential to become an expert user, the majority expressed confidence in their ability to master the approach. In fact, 58.3% fully agreed, and 41.7% partially agreed that the MoT approach is easy to master.

Question	Agree Totally	Agree	Neutral	Disagree	Disagree Totally
The prototype of MoT is easy to use	6	6	0	0	0
MoT's approach is easy to learn	6	6	0	0	0
MoT's approach is easy to master	7	5	0	0	0

Table 5: Result regarding the perception of ease of use.

Regarding perceived utility, Table 6 presents the data collected from participants. All participants agreed that the MoT approach would facilitate the development of CoT applications, with 58.3% fully agreeing and 41.7% partially agreeing. When evaluating MoT's impact on developer productivity, 58.3% of participants fully agreed, while the remaining 41.7% partially agreed. Regarding the potential reduction in delivery time for CoT applications, 50% fully agreed, while the other 50% partially agreed. Regarding the abstraction of IoT components, 50% of participants fully agreed that it reduces the complexity of CoT application development, while the other 50% partially agreed. Similarly, 58.3% of participants partially agreed that abstracting cloud computing components reduces the complexity of environment configurations, while 41.7% fully agreed. These results highlight a strong overall belief in the utility and efficiency of the MoT approach across different aspects of CoT development.

Table 7 presents participants' general perceptions of the MoT approach. All participants agreed that MoT could support CoT application development in the software industry, with 25% strongly agreeing and 75% agreeing. Regarding users with limited technical knowledge of IoT and cloud computing developing a CoT application, 41.7% strongly agreed, 50% agreed, and 8.3% remained neutral. Furthermore, 83.3% of participants intended to use MoT to develop CoT applications, while 16.7% remained neutral. Overall, these findings indicate positive perceptions of MoT's industrial applicability, accessibility, and potential adoption among the evaluated participants.

Therefore, the collected and analyzed data strongly suggest that the MoT approach has significant potential for acceptance among individuals with profiles similar to the

Question	Agree Totally	Agree	Neutral	Disagree	Disagree Totally
The MoT approach would facilitate the development of CoT applications	7	5	0	0	0
MoT's approach would help developer productivity	7	5	0	0	0
MoT approach would reduce the time to deliver a CoT application	6	6	0	0	0
IoT component abstraction reduces development complexity	6	6	0	0	0
Cloud computing component abstraction reduces the complexity of environment configuration	7	5	0	0	0

Table 6: Result regarding perceived utility.

Question	Agree Totally	Agree	Neutral	Disagree	Disagree Totally
The MoT approach could be used for CoT application development in the software industry	3	9	0	0	0
A person with little technical knowledge in IoT development and cloud computing could build a home CoT application	5	6	1	0	0
I would use the MoT approach to create a CoT application	1	9	2	0	0

Table 7: Result regarding general perception.

participants'. The results highlight the approach's usability and its promising real-world application, underscoring its potential for broad adoption and practical implementation.

6 Conclusion and Future Work

This study explored model-driven development approaches for creating IoT applications that harness cloud computing capabilities. To achieve this, we proposed a model-driven approach called MoT, specifically designed to address critical challenges in CoT applications, with a particular focus on managing cloud service heterogeneity. Our research aimed to investigate whether a model-driven approach could streamline the development of native cloud IoT applications with minimal manual intervention and whether individuals with limited IoT experience could successfully create applications using this approach. To evaluate MoT's effectiveness, we conducted a case study and applied the Technology Acceptance Model.

Our initial results showed that the MoT approach effectively facilitates creating CoT applications with minimal manual effort, as shown in the case study. However, the approach currently lacks full functionality for running applications directly in the cloud. Despite this limitation, the TAM-based evaluation yielded positive feedback from study participants. While this is an initial assessment, the results indicate that MoT is perceived as both useful and easy to use, even among individuals with limited IoT knowledge. These findings affirm the approach's potential and suggest promising acceptance among technology professionals, offering a favorable response to the second research question.

As part of our ongoing research agenda, future work will expand MoT beyond its current execution-oriented foundation to incorporate an event-driven architectural style

[Lazzari & Farias 2023] and broader operational coverage. First, we plan to investigate formal concurrency modeling and synchronization mechanisms to better capture parallel execution behaviors, coordination requirements, and potential contention scenarios that arise in Cloud-of-Things environments. Second, we intend to strengthen systematic support for complex multi-device configurations by addressing distributed orchestration, inter-device dependencies, and scalable deployment strategies. Finally, we will explore architectural strategies to better manage heterogeneity across different cloud backends, including resilience and fault-tolerance mechanisms that handle failures, variable service availability, and diverse platform constraints. Together, these research directions aim to mature MoT toward a more comprehensive and robust framework capable of supporting large-scale, heterogeneous, and mission-critical CoT applications. Researchers can also explore ways to enhance MoT and evaluate its effectiveness across diverse contexts. These include: (1) incorporating additional UML diagrams to broaden the modeling capabilities, (2) comparing MoT with traditional IoT development methods to assess its relative advantages, (3) expanding the diversity of participants to include both general users and experienced IoT professionals, (4) the need for a more flexible and configurable transformation engine, (5) mechanisms to support rollback, resilience, and fault handling during deployment, and (6) the design of systematic performance and scalability evaluations. This article opens a clear path for future efforts to formalize optimization objectives, define measurable constraints, evaluate correctness, and introduce quantitative performance baselines.

These improvements would provide deeper insights into MoT's strengths and limitations, bringing the research closer to establishing a solid foundation for advancing Model-Driven Development in IoT application creation. Rather than constituting a definitive study, our work serves as an initial step toward a more comprehensive research agenda in this area.

Ethics Issue

We conducted the research in settings where ethics approval for survey studies was deemed unnecessary because the investigation involved voluntary participation and collected no sensitive information.

Acknowledgement

This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), under Grant No. 312320/2025-6.

References

[Alhaidari et al. 2023] Alhaidari, F., Rahman, A., Zagrouba, R.: ''Cloud of Things: Architecture, Applications and Challenges''; Journal of Ambient Intelligence and Humanized Computing, 14(5), 2023, pp. 5957-5975.

[Aazam et al. 2016] Aazam, M., Huh, E.-N., St-Hilaire, M., Lung, C.-H., Lambadaris, I.: ''Cloud of Things: Integration of IoT with Cloud Computing''; In: Robots and Sensor Clouds, Springer, 2016, pp. 77–94.

[Belguidoum et al. 2022] Belguidoum, M., Gourari, A., Sehili, I.: ''MDMSD4IoT: a Model-driven Microservice Development for IoT Systems''; International Conference on Model and Data Engineering, 2022, pp. 176–189.

[Baldoni et al. 2023] Baldoni, G., Teixeira, R., Guimarães, C., Antunes, M., Gomes, D., Corsaro, A.: ''A Dataflow-oriented Approach for Machine-learning-powered Internet of Things Applications''; Electronics, 12(18), 2023, p. 3940.

[Corradini et al. 2023] Corradini, F., Fedeli, A., Fornari, F., Polini, A., Re, B., and Ruschioni, L.: ''X-IoT: a Model-driven Approach to Support IoT Application Portability across IoT Platforms''; Computing, 2023, Vol. 105, No. 9, pp. 1981-2005.

[D'Avila et al. 2020] D'Avila, L.F., Barbosa, J.L.V., Farias, K.: ''SW-Context: a Model to Improve Developers' Situational Awareness''; IET Software, vol. 14, no. 5, pp. 535–543, 2020.

[Davis 1989] Davis, F. D.: ''Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology''; MIS Quarterly, 1989, pp. 319–340.

[de Oliveira and Lopes 2020] de Oliveira, R. S., Lopes, D.: ''An Approach based on Model Driven Engineering to Support the Development of Web of Things''; Proceedings of the 2020 European Symposium on Software Engineering, 2020, pp. 168–174.

[Farias et al. 2017] Farias, K., Oliveira, T., Gonçales, L. J., Bischoff, V.: ''Modeling Composition of UML Profiles with Alloy''; International Journal of Computer Science and Software Engineering, 6(10), 2017, pp. 223–232.

[Farias et al. 2015] Farias, K., Garcia, A., Whittle, J., von Flach Garcia Chavez, C., Lucena, C.: ''Evaluating the Effort of Composing Design Models: A controlled Experiment''; Software & Systems Modeling, 14(4), 2015, pp. 1349–1365.

[Farias et al. 2019] Farias, K., Cavalcante, T., Gonçales, L. J., Bischoff, V.: ''UML2Merge: A UML Extension for Model Merging''; IET Software, 13(6), 2019, pp. 575–586.

[Fazel and Wainer 2023] Fazel, I. A., Wainer, G.: ''A DEVS-Based Methodology for Simulation and Model-Driven Development of IoT''; International Conference on Simulation Tools and Techniques, 2023, pp. 3–17. Springer.

[Fortas et al. 2022a] Fortas, A., Kerkouche, E., Chaoui, A.: ''Application of MDE in the Development of IoT Systems: A Simulation-Based Approach''; First International Conference on Computer Communications and Intelligent Systems (I3CIS), 2022, pp. 93–98.

[Gubbi et al. 2013] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: ''Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions''; Future Generation Computer Systems, 29(7), 2013, pp. 1645–1660.

[Hassan and Qasha 2021] Hassan, H. A., Qasha, R. P.: ''Toward the Generation of Deployable Distributed IoT System on the Cloud''; IOP Conference Series: Materials Science and Engineering, 2021, Vol. 1088, No. 1, 2021, pp. 1-11.

[Hellal et al. 2024] Hellal, Y. G., Hamel, L., Graiet, M., Balouek, D.: ''A Formal Modeling and Verification Approach for IoT-Cloud Resource-Oriented Applications''; 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2024, pp. 347–356.

[Jiuting and Mohajer 2025] Jiuting, Y., Mohajer, A,: ''Multi-Objective Constellation Optimization and Dynamic Link Utilization for Sustainable Information Delivery Using PD-NOMA Deep Reinforcement Learning''; Wireless Networks, 2025, Vol 31, No. 2, 2025, pp. 1839-1859.

[Kuang et al. 2025] Kuang, S., Zhang, J., and Mohajer, A.: ''Reliable Information Delivery and Dynamic Link Utilization in MANET Cloud Using Deep Reinforcement learning''; Transactions on Emerging Telecommunications Technologies, 2024, Vol. 35, No. 9, pp. 1-24.

[Lazzari & Farias 2023] Lazzari, L., Farias, K.: ''Event-driven Architecture and REST Architectural Style: An exploratory Study on Modularity''; Journal of applied research and technology, 2023, vol. 21, No. 3, pp. 338-351.

[Marangunić & Granić 2015] Marangunić, N., Granić, A.: ''Technology Acceptance Model: A Literature Review from 1986 to 2013''; Universal Access in the Information Society, vol. 14, no. 1, pp. 81–95, 2015.

[Martin 2018] Martin, R. C.: ''Clean Architecture: A Craftsman's Guide to Software Structure and Design''; Pearson Education, Boston, MA, 2018.

[Menzen et al. 2021] Menzen, J., Farias, K., Bischoff, V.: ''Using Biometric Data in Software Engineering: A Systematic Mapping Study''; Behaviour & Information Technology, 2021, Vol. 40, No. 9, pp. 880-902.

[Mohajer et al. 2024] Mohajer, A., Hajipour, J., and Leung, V.: ''Dynamic Offloading in Mobile Edge Computing with Traffic-aware Network Slicing and Adaptive TD3 Strategy''; IEEE Communications Letters, 2024, Vol. 29, No. 1, January, pp. 95-99.

[Oliveira et al. 2008] Oliveira, K., Breitman, K., Oliveira, T.: ''A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap''; Journal of Universal Computer Science, 2008, Vol. 15, No. 11, pp. 2225-2253.

[OMG 2017] OMG: ''Unified Modeling Language Specification''; Version 2.5.1, 2017.

[Panayiotou et al. 2025] Panayiotou, K., Tsardoulias, E., and Symeonidis, A.: ''LocSys: A Low-Code Paradigm for the Development of Cyber-Physical Applications''; Sensors, 2025, Vol. 25, No. 3951, pp. 1-25.

[Richards & Ford 2020] Richards, M., and Ford, N.: ''Fundamentals of Software Architecture: An Engineering Approach''; O'Reilly Media, 2020.

[Rubert and Farias 2022] Rubert, M., Farias, K.: ''On the Effects of Continuous Delivery on Code Quality: A Case Study in Industry''; Computer Standards & Interfaces, 81, 2022, pp. 1-20.

[Sree et al. 2022] Sree, N. S., Lakshmi, P. P., Mary, S. P.: ''For Mobile Services in the Cloud of Things: A Model-driven Development Pattern''; International Journal of Cloud Computing, 11(5–6), 2022, pp. 511–517.

[Suhailam et al. 2024] Suhailam, P., Yerolla, R., Besta, C. S.: ''Cloud of Things: Architecture and Industrial Applications''; In: Cloud of Things, Chapman and Hall/CRC, 2024, pp. 1-17.

[Terracher et al. 2024] Terracher, A., Kocik, R., Hamouche, R., Rachedi, A., Bourouina, T.: ''Model-Based Approach for Efficient Data Exploitation Process in IoT Systems''; 2024 International Wireless Communications and Mobile Computing (IWCMC), IEEE, 2024, pp. 885-890.

[Urdangarin et al. 2021] Urdangarin, R., Farias, K., Barbosa, J.: ''Mon4Aware: A Multi-Objective and Context-aware Approach to Decompose Monolithic Applications''; XVII Brazilian Symposium on Information Systems, 2021, pp. 1-9.

[Vieira & Farias 2025] Vieira, R., Farias, K.: ''A Tool-Supported Approach to Integrate Cognitive Indicators into the Visual Studio Code''; Journal of Universal Computer Science, vol. 31, no. 7, p. 683, 2025.

[Yadav et al. 2024] Yadav, D., Yadav, A., Singh, S., et al.: ''Concerns, Threats, and Ethical Considerations in Using Cloud of Things''; In: Cloud of Things, Chapman and Hall/CRC, 2024, pp. 345-364.

[Wohlin et al. 2012] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., et al.: ''Experimentation in Software Engineering''; Springer, Berlin, 2012, Vol. 236.