IoTCRAFT: Toward a Declarative Approach for IoT System Specification and Deployment

João Cardoso[1], Pedro Salgueiro[1], and Teresa Gonçalves[1]

Centro Algoritmi / LASI, VISTA Lab
Universidade de Évora
Évora, Portugal
{joao.pedro.cardoso, pds, tcg}@uevora.pt

Abstract. Developing and deploying IoT systems remains a complex task, often involving the manual integration of heterogeneous components for implementing data pipelines, data storage, visualization, and machine learning workflows. This paper proposes IoTCRAFT, a declarative platform designed to define high-level specifications and automate the deployment of Internet of Things (IoT) systems. The platform is intended to support complete data processing pipelines, including data ingestion, validation, normalization, storage, and visualization, as well as the training and inference of machine learning models. By using a high-level specification to describe system behavior, the platform's code generation and orchestration engine can automatically provision the required infrastructure using container-based technologies. This approach aims to reduce the need for manual configuration and simplifies the deployment of IoT systems.

Keywords: Declarative Programming; Orchestration; Specification; IoT

1 Introduction

The rapid growth of Internet of Things (IoT) technologies has led to a growing number of intelligent systems that collect, process, store and analyze data from distributed sensors and devices [7, 22]. These systems are now widely deployed, supporting applications, for example, in environmental monitoring and smart cities.

However, developing such systems remains a complex, manual and error-prone process [1]. Developers must integrate heterogeneous components for data ingestion (handling stream or batch processing), storage solutions, visualization dashboards, and ML (Machine Learning), often using low-level configuration files and ad hoc scripts. Maintaining these systems becomes even more challenging when aiming for scalability and reproducibility of this systems.

As IoT deployments grow in scale, they produce vast volumes of heterogeneous data that need to be efficiently ingested, processed, stored, and made accessible to end users. Building and managing the pipelines to handle this full

process, from raw sensor outputs to final analytic results, typically demands manual effort and custom integration of components [1].

Despite the maturity of many individual IoT technologies, current development practices still lack of high-level abstractions for end-to-end specification and deployment of IoT systems [2]. Moreover, declarative techniques that have proven successful in domains like cloud orchestration and Infrastructure-as-Code remain only partially explored in the IoT domain, particularly for orchestrating real-time data streams and integrated ML components [10].

To address these challenges, this paper propose IoTCRAFT, a declarative platform for specifying and deploying IoT systems. This platform is being designed to leverage high-level, technology-agnostic specifications to automate the full lifecycle of IoT data pipelines, including data normalization, validation, data analytic processing, and ML workflows. The platform seeks to unify system specification, deployment, monitoring, and adaptation by incorporating principles from Development and Operations (DevOps) and Machine Learning Operations (MLOps) into its design. By minimizing the need for manual configuration, IoTCRAFT bridges the gap between high-level system modeling and low-level deployment details, aiming to simplify IoT system development and maintenance.

As the work is still in its early stages, the proposed architecture, and all the components are currently being designed and explored. This paper outlines the initial vision of the platform and highlights the foundational directions that will guide its future development.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 presents the problem statement. Section 4 introduces the architecture of the IoTCRAFT platform. Section 5 describes the validation scenario and use case and Section 6 concludes the paper.

2 Related Work

The development of intelligent IoT systems involves several interconnected aspects, such as device connectivity, data pipelines, storage, deployment infrastructure and machine learning workflows. Recent work has shown that declarative approaches can help reduce complexity by allowing high-level specification of system behavior, simplifying configuration and improving reproducibility.

This section reviews relevant research and tools related to declarative approaches and automation in the IoT. It highlights current strengths and limitations, and identifies opportunities for a unified platform that combines all the mentioned components.

2.1 Declarative Principles and Motivation

IoT systems are complex due to the wide variety of devices involved, the dynamic of the data, and the diversity of deployment environments [2]. Recent research

highlights that developing and maintaining these systems presents major challenges in scalability, interoperability, and maintainability.

To address this complexity, many research efforts support declarative approaches, where developers describe the desired outcomes or system state, rather than the detailed steps needed to achieve them. By increasing the level of abstraction, declarative paradigms allow developers to define what the system should do, while the platform determines how to implement those definitions. This approach helps reduce the need to manually manage low-level details and also facilitate maintenance.

One important motivation for using declarative methods in IoT is to improve reliability and safety at scale. In large and heterogeneous systems, manually scripting every operation becomes impractical and error-prone. In work [15], the authors propose a declarative configuration API for IoT operations, showing that it helps prevent system faults and reduces the operational complexity. By validating high-level configurations instead of sequences, it becomes easier to avoid unintended system behaviors.

Declarative specifications also support formal reasoning and validation. For example, model-driven or goal-oriented IoT frameworks allow constraints and policies to be checked before deployment, helping to ensure correctness [2].

In summary, the benefits of declarative paradigms in IoT lie in their ability to manage complexity through abstraction, addressing the heterogeneity with unified specifications, and improving scalability and maintainability by automating low-level execution.

2.2 Declarative Programming and DSLs for IoT

Declarative programming focuses on specifying *what* a system should do, rather than detailing the *how*. By abstracting the execution logic, it allows developers to define high-level goals while delegating implementation details to the system. This can reduce development and deployment complexity.

In general, one of the most common declarative approaches is the use of Domain-Specific Languages (DSLs). DSLs are tailored languages that allow developers to express application logic in a concise way. In IoT systems, DSLs have been designed to, for example, specify device behavior, communication structures, and system policies.

Recent studies confirm a growing number of DSLs and modeling languages specifically designed for IoT. For instance, the work presented in [2] analyze 32 DSLs that target different aspects of IoT systems, from node-centric programming models to architecture description languages. Despite the variety of approaches, most of them share an objective, that is enabling platform independent specification of system behavior and structure.

One example is Task-Oriented Programming (TOP), a declarative paradigm where developers define tasks and their data dependencies. The runtime is then responsible for orchestrating the execution across distributed nodes [18]. This simplifies coordination without exposing details such as message passing or synchronization.

Another line of work is the use of actor-based models, where applications are composed of autonomous components (actors) that communicate via asynchronous messages. For instance, the work described in [1] presents a formal actor model for resource-constrained IoT services and this approach promotes modularity and fault isolation, while enabling formal reasoning over concurrent behaviors and deployment constraints.

Some DSLs focus on architecture level modeling. MontiThings [16] is a work, where the model-driven framework adopts a component-and-connector (C&C) approach. It enables developers to define high-level diagrams describing data flow and component interactions. From these models, the framework generates reliable distributed code, including communication and deployment logic. Similarly, the tool ThingML provides platform-independent specifications that can be compiled into code for different embedded and cloud platforms [11].

Despite their strengths, most existing DSLs only cover parts of the development lifecycle. While some focus on state machines or rule-based programming, others handle only data flow or component deployment. As noted in [2], no single DSL currently addresses all relevant dimensions of IoT application design, leading to a trend toward hybrid or multi paradigm modeling languages.

In addition to external DSLs, many frameworks rely on declarative configurations or internal DSLs. Declarative configurations, often written in YAML or JSON, allow users to define pipelines or services without writing code. For example, EdgeX Foundry lets users declare data processing flows using YAML, which the framework then interprets and executes at runtime.

These approaches offer benefits similar to Infrastructure-as-Code, however, they are limited by the expressiveness of the configuration schema, where users can only configure what was anticipated by the framework designers.

To overcome these limitations, some systems adopt internal DSLs built with fluent APIs in general-purpose languages. This offers more flexibility, but can blur the line between declarative and imperative programming [9].

In summary, the design of DSLs for intelligent IoT systems must balance expressiveness with usability. The goal is to provide high-level abstractions that hide unnecessary complexity (e.g., devices, network protocols and deployment mechanisms) while remaining accessible to developers. Some platforms adopt a hybrid approach, combining visual modeling with scripting, to provide both abstraction and flexibility [9].

Event-Driven and Actor-Based Models. IoT applications frequently produce high volumes of events from sensors and distributed nodes. As a result, event-driven architectures are widely adopted in this domain. These architectures allow components to interact asynchronously through events or messages, enabling loose coupling between data producers and consumers.

This reactive model improves scalability and response, allowing independent components to process data in parallel. In combination, the actor model offers an abstraction for building distributed IoT systems. Actors encapsulate state and behavior and interact exclusively through asynchronous message passing.

This model simplifies concurrency management and improves fault tolerance for systems running on constrained or intermittently connected devices.

For example, the work [1] presents an actor-based model designed for verifying and deploying resource-bounded IoT services. Other works, such as [26], use high-level models where events propagate through operator graphs mapped to actors. This allows developers to describe complex logic using reactive and modular structures.

Together, event-driven and actor-based paradigms provide a solid foundation for designing IoT systems. They are also well aligned with declarative programming principles, as they separate communication concerns from core logic, allowing the system to react automatically to changes.

2.3 Declarative Deployment and Infrastructure Automation

Declarative deployment is in some way related to the concept of Infrastructure as Code (IaC) and has revolutionized the way cloud infrastructures are managed. Instead of writing imperative scripts or performing manual steps, developers define the desired target state of the infrastructure (e.g., containers, services, network policies), and orchestration tools ensure that the system converges to this state automatically.

Tools such as Kubernetes [17] and Terraform [24] exemplify this paradigm. They interpret high-level specifications and manage resources by reconciling the actual system state with the declared configuration. This declarative infrastructure significantly improves reproducibility, reduces manual effort, and lowers the risk of configuration errors.

Deployment platforms typically support declarative models in which system components, configurations, and dependencies are described as resource graphs [23]. In the IoT domain, this model has been extended to support hybrid topologies that span cloud, fog, and edge environments. These distributed infrastructures require flexible deployment strategies to address constraints such as latency, bandwidth, energy, or availability.

Research proposes frameworks that enable runtime reconfiguration of deployments based on events such as load variation or node failure. These systems are guided by high-level declarative policies, allowing the platform to automatically adapt the deployment without developer intervention [9]. This type of automation is particularly important for managing large-scale IoT infrastructures, where services may need to be dynamically moved or replicated to maintain performance and availability.

2.4 Declarative ML Pipelines in IoT

Integrating Machine Learning into IoT systems introduces some challenges, including the handling of large scale, heterogeneous data streams and also the orchestration of resource for training and inference workflows.

Declarative ML pipelines aim to address these challenges by abstracting the entire machine learning lifecycle, from data preprocessing to model training and

evaluation. Tools such as TensorFlow Extended (TFX) and Kubeflow Pipelines allow's to define workflows using high-level configuration files (e.g., YAML), while the underlying system manages task scheduling and data flow.

In IoT environments, data is often distributed across the cloud, and this creates additional requirements for pipeline orchestration across heterogeneous and geographically dispersed nodes. To support this, recent frameworks have been proposed that extend declarative ML pipelines across the cloud–edge continuum.

For example, the work [3] introduce *Zenoh-Flow*, a decentralized framework that allows developers to declaratively specify ML pipelines as dataflow graphs. Each node in the graph represents an operator (e.g., filtering, inference), along with non-functional constraints such as latency or resource requirements. The system then deploys these operators dynamically across available nodes, optimizing for context and constraints.

Similarly, work [4] explore adaptive placement of ML inference stages based on runtime conditions. Their system uses unified declarative specifications to reconfigure pipeline deployment automatically, improving responsiveness and resource utilization.

By abstracting operational complexity, declarative ML pipelines promote automation and flexibility for deploying intelligent IoT systems at scale across diverse infrastructures.

DevOps/MLOps for Declarative IoT. The reliable development and operation of intelligent IoT systems require the integration of DevOps practices with Machine Learning Operations (MLOps). Traditional DevOps focuses on automating software development, testing, deployment, and monitoring. When ML is introduced, this extends to model training, validation, versioning, deployment, and performance monitoring.

In IoT, ML scenarios add additional challenges, including data drift, model degradation, and the need to deploy models to heterogeneous and often resource constrained devices. Furthermore, ML models may require frequent retraining and re-deployment to remain effective in changing environments.

Surveys such as [21] propose taxonomies and best practices for managing the ML lifecycle in IoT systems. They emphasize the importance of automation and reproducibility. Declarative specifications of infrastructure and ML workflows are seen as a key enabler, helping to ensure consistency across deployments.

Platforms such as Edge Impulse [12] provide integrated MLOps support tailored to IoT. These platforms allows developers to collect sensor data, build and train models, and deploy updates to edge devices, all through high-level declarative interfaces. They also support model versioning, testing and performance tracking.

In summary, the convergence of DevOps, MLOps, and declarative practices aims to enable a fully automated and adaptive lifecycle, from initial design to runtime evolution, while maintaining alignment with application goals and operational constraints.

2.5 Declarative Architectures and Research Prototypes

This section presents recent research prototypes/platforms that apply declarative principles to the development and deployment of IoT systems. These solutions typically rely on domain-specific languages (DSLs), orchestration engines, or low-code tools to support high-level specification of architectures and data workflows. They all share the common goal of reducing complexity by allowing developers to describe system structure and behavior declaratively, while the underlying platform manages deployment and integration across cloud infrastructure.

AutoIoT (2020). AutoIoT is a declarative framework designed to simplify the development of IoT applications by using model-driven engineering techniques [19]. Developers define IoT systems through a high-level JSON model that describes devices, data flows, and user interfaces. From this model, the AutoIoT automatically generates server-side applications that include data ingestion endpoints (e.g., MQTT or REST), storage, and basic dashboards. The goal is to reduce complexity and make IoT backend development accessible to non experts. In a user study with 54 participants, AutoIoT showed high usability and effectiveness [19].

However, AutoIoT has some limitations, it focuses mainly on generating centralized server applications and assumes that data acquisition from devices is handled externally. It does not support distributed deployment or integration of machine learning. Extending the framework requires creating custom code generators, which can be difficult for non-specialists.

IoTCRAFT builds on the declarative principles of AutoIoT, but with a DSL and takes a broader approach. It aims to support the full IoT pipeline, from data ingestion to machine learning and visualization, using a unified high-level specification. Also, unlike AutoIoT, it integrates DevOps/MLOps practices to automate deployment, monitoring, and model lifecycle management.

MontiThings (2022) - Model-Driven IoT Platform. MontiThings [16] is a model-driven platform that provides end-to-end support for developing and deploying IoT applications. It introduces a group of DSLs to model IoT systems as component-and-connector (C&C) architectures, enabling a clear separation between high-level behavior and low-level device details.

Applications are defined as interconnected components with typed ports for device I/O. Component behavior can be specified using state machines, an imperative DSL tailored to IoT, or embedded code. From these models, MontiThings automatically generates distributed *C++* microservices and manages deployment to heterogeneous IoT and edge devices.

A model-driven "App Store" mechanism supports device specific configuration, decoupling software from hardware to simplify redeployment. Fault tolerance is built into the runtime through automated error-handling code injection and diagnostic tools such as logging and record/replay mechanisms.

MontiThings allows developers to focus on application logic while abstracting communication, deployment, and reliability concerns. Although analytics and ML components can be integrated into the models, this process remains manual.

In contrast, the platform proposed in this paper aims to support end-to-end declarative specification of intelligent IoT systems, including sensor data ingestion, real-time visualization, and machine learning workflows, along with DevOps and MLOps practices. While both approaches share goals of abstraction and automation, MontiThings focuses on embedded device coordination, whereas this research targets data-centric, pipeline-oriented systems across heterogeneous infrastructures.

GeneSIS (2020) - Declarative Deployment for Smart IoT Systems. GeneSIS [10] is a model-driven framework designed to simplify the development and continuous delivery of smart IoT systems across IoT, edge, and cloud layers. It introduces a DSL that enables developers to declaratively specify system topology, including devices, services, pipelines, and their deployment locations, along with non-functional properties such as security constraints.

The orchestration engine follows a configurable runtime approach, where deployment models are interpreted at *runtime* to provision and configure components across heterogeneous infrastructure, injecting security mechanisms such as encryption and access control where required. A cloud-based deployment manager coordinates the process, while local agents on IoT and edge nodes handle on-site installation, including support for intermittent connectivity.

After deployment, GeneSIS maintains model–runtime consistency and monitors the system and performs reconfiguration when faults are detected or when the model is updated.

This framework supports DevSecOps practices by enabling developers to model system configurations once and generate secure, reproducible deployments. However, its focus is limited to deployment-level concerns.

The platform proposed in this work aims to generalize the declarative approach beyond deployment, covering ingestion, processing, visualization, and ML integration, and to incorporate lifecycle automation through DevOps and MLOps principles. GeneSIS offers foundational mechanisms for secure orchestration, but lacks native support for data analytics and intelligent adaptation workflows.

MLIoT (2021) - End-to-End IoT Data Analytics Pipeline. MLIoT [5] is a platform specifically designed to support the full machine learning lifecycle in IoT applications, from data ingestion to edge deployment and model adaptation. The architecture is divided into two layers, a Device-Selection Layer and a Training-Serving Layer.

The Device-Selection Layer manages a registry of IoT and edge nodes, including their computational capabilities. Based on declarative user defined policies, for example, minimizing latency or maximizing inference accuracy, it selects the most appropriate node to run each ML task.

The Training-Serving Layer automates the entire pipeline, ingesting data, performing parallel model training, selecting the best model, and deploying it to the appropriate node for inference. It also continuously monitors performance metrics and environmental conditions, triggering retraining or redeployment when necessary.

MLIoT allows developers to define high-level goals (e.g., accuracy or resource constraints) while the platform manages orchestration and adaptation. It has been shown to achieve effective trade-offs between accuracy and efficiency in diverse IoT scenarios by dynamically adjusting model placement and updating inference components.

Unlike platforms focused on infrastructure deployment, MLIoT emphasizes intelligent, adaptive analytics. However, it does not generalize to other aspects of IoT system specification, such as infrastructure orchestration, data, or visualization workflows. The platform proposed in this paper seeks to unify these dimensions under a single declarative specification model.

ThingML (2016) - Things Modeling Language. ThingML is a domain-specific modeling language and code generation framework developed to simplify the creation of IoT applications through high-level, platform-independent specifications. The central concept is to allow developers to model device behaviors, communications, and deployment configurations using a unified declarative syntax, while the ThingML compiler automatically generates efficient source code tailored to various target platforms, such as C/C++ for microcontrollers or JavaScript for gateways and web applications [11].

A ThingML model defines components called *Things*, each encapsulating internal logic through hierarchical state machines, along with *Ports* and *Messages* to handle inter-component communication. Developers can describe complete systems using reusable components and configuration blocks that declare how instances are connected and interact. This model-driven approach enables a single specification to be compiled into firmware for embedded devices and coordination code for edge or cloud components, fostering consistency and reuse across heterogeneous IoT environments.

The low-code of ThingML stems from its declarative syntax rather than implementing device logic manually in low-level languages, developers specify state transitions, actions, and data flow in a compact, structured format. The framework's customizable code generators then produce readable and efficient code, with options for injecting platform-specific instructions when needed. Functionally, ThingML supports synchronous/asynchronous messaging, event-driven behaviors, and includes support for a variety of hardware drivers and network protocols.

Despite its advantages, ThingML presents some challenges. Its specialized modeling paradigm requires developers to learn new concepts and tools, and also maintaining both generated and handwritten code in parallel requires care to avoid conflicts during regeneration. Furthermore, although performance is generally optimized, fine tuning may still be necessary for resource-constrained

environments. While ThingML remains a tool with limited community support, it serves as an example of how model-driven engineering can significantly reduce development effort and bring declarative techniques to the embedded IoT domain [11].

Comparison of Related Platforms with IoTCRAFT. Table 1 summarizes the key features of the four reviewed platforms in comparison with the proposed approach. While MontiThings and GeneSIS primarily target the declarative modeling and deployment of IoT architectures, they do not provide native support for data components such as analytics, dashboards, or machine learning pipelines.

MLIoT, on the other side, focuses on the automated management of ML workflows in IoT scenarios, offering dynamic model deployment and retraining capabilities. However, it lacks high-level modeling support and does not abstract the broader infrastructure and orchestration aspects.

In contrast, the proposed platform (IoTCRAFT) aims to unify the declarative specification of the entire IoT stack, including data ingestion, stream and batch processing, visualization, and machine learning. Additionally, it incorporates DevOps and MLOps practices to support reproducibility, versioning, and lifecycle automation. This unified approach fills a current gap by covering the full IoT stack.

Table 1. Comparison of related platforms with the proposed approach.

Feature	MontiThings	GeneSIS	MLIoT	AutoIoT	ThingML	IoTCRAFT*
Declarative modeling	✓	✓	✗	✓	✓	✓
Sensor data ingestion	✓	✗	✓	✓	✗	✓
Data processing	partial	✗	✓	✗	✗	✓
Visualization	✗	✗	✗	partial	✗	✓
ML integration	✗	✗	✓	✗	✗	✓
Automated deployment	✓	✓	partial	✗	partial	✓
DevOps/MLOps support	✗	✗	partial	✗	✗	✓

* IoTCRAFT features represent planned capabilities. The platform is currently at a conceptual stage.

Open-Source Tools Several open-source platforms adopt declarative or low-code paradigms to streamline the development of IoT solutions, such as Node-RED [20], ThingsBoard [25], Apache StreamPipes [14], EdgeX Foundry [8], and FogFlow [6]. These tools provide graphical or flow-based interfaces for defining data pipelines, dashboards, and basic edge deployments, lowering the entry barrier for IoT development. However, each of them typically targets a specific layer of the stack, such as ingestion, streaming, or orchestration, and lacks comprehensive support for full pipeline specification, especially in integrating machine learning workflows and DevOps/MLOps practices. As such, they offer partial

solutions that require manual integration of components and configurations, reinforcing the need for an end-to-end platform like IoTCRAFT.

3 Problem Statement

The development of intelligent IoT systems presents both conceptual and practical challenges. On one hand, there is a need to define expressive, reusable models that hide technological complexity while preserving control and flexibility. On the other, engineering challenges arise when deploying and managing distributed components, data pipelines, and machine learning workflows in real-world environments. These systems typically require the integration of many components, such as data ingestion pipelines, storage solutions, visualization tools, and machine learning (ML) services, across heterogeneous infrastructures. Managing such complexity often leads to low-level, error-prone implementations that are hard to maintain, adapt, or reuse.

As highlighted in the related work, existing solutions usually address isolated stages of the IoT pipeline or require extensive manual configuration and orchestration. There is a clear lack of integrated platforms that enable the high-level specification and automated deployment of complete IoT-ML systems (i.e., IoT systems integrating machine learning components). This gap hinders reuse, scalability, and developer productivity.

Declarative programming offers a promising alternative by allowing developers to focus on what the system should do, rather than how to implement it. To explore this potential, this paper addresses this gap by exploring the potential of declarative programming for describing and deploying complete IoT systems. A unified, high-level specification language combined with automated orchestration mechanisms can significantly reduce the complexity of these systems. This direction underpins the design of the IoTCRAFT platform, which is currently under development and presented in the following sections.

This work is still in its early stages and focuses on formulating a clear vision for the platform and identifying the foundational mechanisms required to support its implementation. The architectural design, as well as the development of the domain-specific language and orchestration engine, constitute the next steps of the research.

4 IoTCRAFT Architecture

As this research is still in its early stages, this section presents a high-level conceptual view of the IoTCRAFT architecture. It is designed to support the declarative specification and automated orchestration of complete IoT data pipelines, with a focus on abstraction, automation, and modular deployment.

The architecture, illustrated in Figure 1, is organized into three main layers. At the top, the Declarative Layer introduces a high-level specification language (currently under development) that enables users to describe the structure and

behavior of IoT systems. This includes the definition of data sources, validation rules, storage requirements, visualization preferences, and machine learning tasks. The goal is to provide a reusable abstraction that hides low-level configuration details.

The Interpretation and Orchestration Layer bridges the gap between the declarative model and the execution infrastructure. It includes a code generation engine that translates the specification into configuration files and deployment descriptors, and an orchestration engine that provisions the required components using container-based technologies. This encompasses ingestion services (e.g., MQTT or HTTP connectors), validation modules, storage systems, dashboards for data visualization, and machine learning components.

Finally, the Infrastructure Layer comprises the actual services that implement the pipeline. Each stage (ingestion, validation, storage, visualization, ML) is deployed using modular components. This flexibility allows the platform to adapt easily to different IoT use cases.

The layered architecture promotes separation of concerns while ensuring integration between components. Figure 1 illustrates the main components and their interactions.

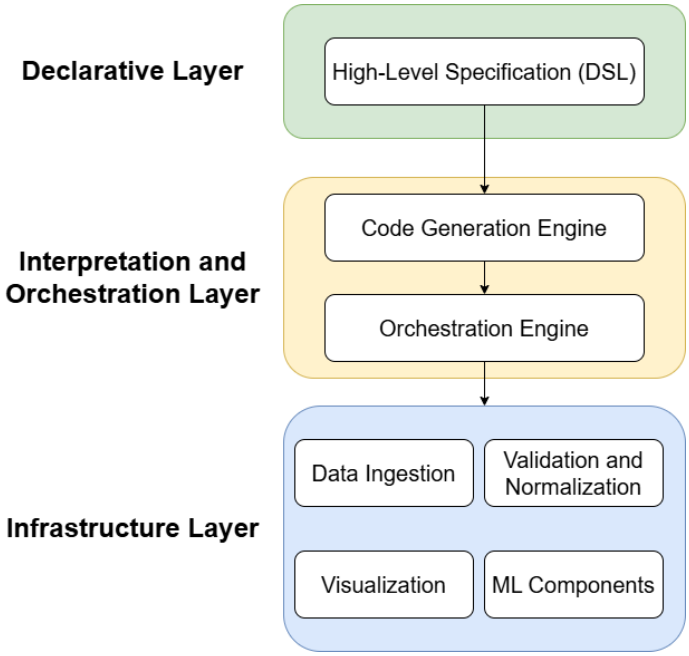

Fig. 1. Conceptual architecture of the IoTCRAFT platform.

5 Validation Scenario and Use Case

To assess the feasibility and benefits of the proposed architecture, IoTCRAFT will be validated using requirements and real sensor data from the SenForFire project [13], a research initiative focused on forest fire prevention through the deployment of low-cost Wireless Sensor Networks (WSNs). These WSNs monitor environmental variables such as temperature, humidity, soil moisture, and air quality across multiple remote and forested locations.

The current implementation of SenForFire requires manual configuration of several system components, including data ingestion services, time-series storage, dashboards, and machine learning workflows. This makes it an ideal scenario to evaluate the potential of IoTCRAFT to reduce complexity and automate the deployment process in a real-world scenario.

The validation strategy involves describing the SenForFire system through a high-level declarative specification and assessing whether the platform can automatically generate and orchestrate the necessary infrastructure. This includes setting up ingestion services, validating and normalizing incoming sensor data, storing it in a time-series database, visualizing it, and executing ML inference tasks.

Although the validation process is still far, this use case provides a good oportunity in a realistic and complex environment to test the expressiveness of the DSL and the automation capabilities of the orchestration engine. It will also offer critical feedback to support the continued development of the platform.

6 Conclusion and Future Work

This paper proposes IoTCRAFT, a novel declarative platform designed to simplify the specification and deployment of IoT systems. By enabling the definition of complete data pipelines through high-level specifications, the platform aims to reduce manual configuration, increase reusability, and promote adaptability across diverse application scenarios.

As highlighted in the related work, there is currently no integrated solution that offers a unified, declarative approach to modeling and deploying the full lifecycle of IoT-ML systems. Existing tools typically address isolated stages of the pipeline or require significant manual configuration and orchestration. IoTCRAFT is conceived to address this gap, offering an end-to-end platform capable of automating complex IoT deployments using a high-level, platform-agnostic specification.

The work is still in an early stage, and all core components are under development. Future efforts will focus on designing the architecture, implementing the required modules, and validating the platform using real-world data collected from the SenForFire project. This use case will serve to assess the feasibility, expressiveness, and automation capabilities of the proposed approach in a realistic deployment scenario.

Acknowledgements

This work has received funding from the Program Interreg-Sudoe of the European Union under grant agreement S1/1.1/E0040 (SenForFire).

References

[1] Ahmed Abdelmoamen Ahmed. "An Actor-Based Formal Model and Runtime Environment for Resource-Bounded IoT Services". In: *Algorithms* 15.11 (2022), p. 390. DOI: 10.3390/a15110390.

[2] Sadik Arslan, Mert Ozkaya, and Geylani Kardas. "Modeling Languages for Internet of Things (IoT) Applications: A Comparative Analysis Study". In: *Mathematics* 11.5 (2023), p. 1263. DOI: 10.3390/math11051263.

[3] Gabriele Baldoni et al. "A Dataflow-Oriented Approach for Machine-Learning-Powered Internet of Things Applications". In: *Electronics* 12.18 (2023), p. 3940. DOI: 10.3390/electronics12183940.

[4] Karolina Bogacka et al. "Flexible Deployment of Machine Learning Inference Pipelines in the Cloud–Edge–IoT Continuum". In: *Electronics* 13.10 (2024), p. 1888. DOI: 10.3390/electronics13101888.

[5] Sudershan Boovaraghavan et al. "MLIoT: An End-to-End Machine Learning System for the Internet-of-Things". In: *Proceedings of the International Conference on Internet-of-Things Design and Implementation (IoTDI)*. New York, NY, USA: ACM, 2021, pp. 169–181. DOI: 10.1145/3450268.3453522.

[6] B. Cheng et al. "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities". In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 696–707. DOI: 10.1109/JIOT.2017.2747214.

[7] Amit Choudhary. "Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions". In: *Discover Internet of Things* 4.31 (2024). DOI: 10.1007/s44200-024-00128-8. URL: https://link.springer.com/article/10.1007/s43926-024-00084-3.

[8] EdgeX Foundry. *EdgeX Foundry Platform Overview*. Accessed: 2025-05-26. 2024. URL: https://www.edgexfoundry.org/software/platform/.

[9] Nicolas Farabegoli et al. "Scalability through Pulverisation: Declarative deployment reconfiguration at runtime". In: *Future Generation Computer Systems* 161 (2024), pp. 545–558. DOI: 10.1016/j.future.2024.07.042.

[10] Nicolas Ferry et al. "Continuous Deployment of Trustworthy Smart IoT Systems". In: *Journal of Object Technology* 19.2 (July 2020), 16:1–23. DOI: 10.5381/jot.2020.19.2.a16.

[11] N. Harrand et al. "ThingML: A Language and Code Generation Framework for Heterogeneous Targets". In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016)*. New York, NY, USA: ACM, 2016, pp. 125–135. DOI: 10.1145/2976767.2976812.

[12] Shawn Hymel et al. "Edge Impulse: An MLOps Platform for Tiny Machine Learning". In: *Proc. 5th Conf. on Machine Learning and Systems (MLSys)*. Available as arXiv:2212.03332. 2023.

[13] Interreg Sudoe. *SenForFire*. Accessed: February 16, 2025. 2025. URL: https://interreg-sudoe.eu/en/proyecto-interreg/senforfire/.

[14] M. Jacoby et al. "An Approach for Realizing Hybrid Digital Twins Using Asset Administration Shells and Apache StreamPipes". In: *Information* 12.6 (2021), p. 217. DOI: 10.3390/info12060217.

[15] Rajat Kandoi and Klaus Hartke. "Operating large-scale IoT systems through declarative configuration APIs". In: *Proc. of the 2021 Workshop on Descriptive Approaches to IoT Security, Network, and Application Configuration (DAI-SNAC@CoNEXT)*. ACM, 2021, pp. 22–25.

[16] J"org Christian Kirchhof et al. "MontiThings: Model-Driven Development and Deployment of Reliable IoT Applications". In: *Journal of Systems and Software* 183 (2022), p. 111087. DOI: 10.1016/j.jss.2021.111087.

[17] *Kubernetes*. https://kubernetes.io/docs/. Accessed: July 25, 2025.

[18] Mart Lubbers, Pieter Koopman, and Rinus Plasmeijer. "Writing Internet of Things Applications with Task Oriented Programming". In: *Composability, Comprehensibility and Correctness of Working Software*. Ed. by Zolt'an Porkol'ab and Vikt'oria Zs'ok. Cham: Springer International Publishing, 2023, pp. 3–52. DOI: 10.1007/978-3-031-42833-3_1.

[19] Thiago Nepomuceno et al. "AutoIoT: A Declarative Framework for Building IoT Applications". In: *Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing (SAC 2020)*. Brno, Czech Republic: ACM, 2020, pp. 719–728. DOI: 10.1145/3341105.3373873.

[20] I.-V. Nițulescu and A. Korodi. "Supervisory Control and Data Acquisition Approach in Node-RED: Application and Discussions". In: *IoT* 1.1 (2020), pp. 76–91. DOI: 10.3390/iot1010005.

[21] Bin Qian et al. "Orchestrating the Development Lifecycle of Machine Learning-Based IoT Applications: A Taxonomy and Survey". In: *ACM Computing Surveys* 53.4 (2020), pp. 1–47. DOI: 10.1145/3398020.

[22] Kalle Sorri, Navonil Mustafee, and Marko Seppänen. "Revisiting IoT definitions: A framework towards comprehensive use". In: *Technological Forecasting and Social Change* 179 (2022). DOI: 10.1016/j.techfore.2022.121623. URL: https://doi.org/10.1016/j.techfore.2022.121623.

[23] Miles St"otzner et al. "Modeling Different Deployment Variants of a Composite Application in a Single Declarative Deployment Model". In: *Algorithms* 15.10 (2022), p. 382. DOI: 10.3390/a15100382.

[24] *Terraform*. https://www.terraform.io/. Accessed: July 25, 2025.

[25] ThingsBoard Inc. *ThingsBoard Microservices Architecture*. Accessed: 2025-05-26. 2024. URL: https://thingsboard.io/docs/reference/msa/.

[26] Yang Zhang and Jun-Liang Chen. "Declarative Construction of Distributed Event-Driven IoT Services Based on IoT Resource Models". In: *IEEE Transactions on Services Computing* 14.1 (2021), pp. 125–140. DOI: 10.1109/TSC.2018.2883449.