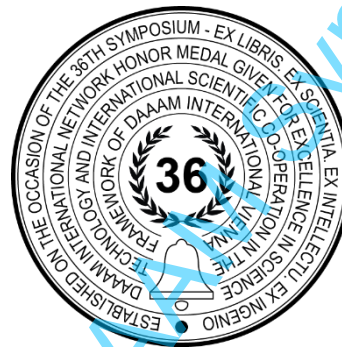


AN EXTENSIBLE MULTI-AGENT ARCHITECTURE FOR MANAGING ORGANIZATIONAL INTERNET OF THINGS SYSTEMS BASED ON THE MODEL CONTEXT PROTOCOL

Borna Kadežabek, Snježana Križanić & Neven Vrčec



This Publication has to be referred as: Kadežabek, [B]orna; Križanić, S[nježana] & Vrčec, N[even] (2026). An Extensible Multi-Agent Architecture for Managing Organizational Internet of Things Systems Based on the Model Context Protocol, Proceedings of the 37th DAAAM International Symposium, pp.xxxx-xxxx, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-xx-x, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/37th.daaam.proceedings.xxx

Abstract

Organizations increasingly rely on heterogeneous Internet of Things (IoT) systems, yet existing research on integrating Large Language Models (LLMs) with IoT infrastructure via the Model Context Protocol (MCP) primarily addresses a narrow, single-domain application rather than multi-agent coordination at the organizational level. This research proposes an extensible software-hardware architecture for managing organizational IoT systems, where MCP serves as a central integration layer between LLM-based Agents and physical devices. The proposed architecture extends the existing decoupled IoT-MCP design with an Agent Orchestrator which instantiates one or more Agent sessions for a given task and retains central authority over their coordination with a generic tool design based on a device registry. In this design, tools are defined by function rather than by specific physical devices, allowing devices of already supported types to be integrated into the system by adding them to the registry, without modifying existing tools or logic. Each design decision is justified by its implications for extensibility and coordination, including constraints introduced by the architecture itself. A minimal implementation using simulated hardware is provided to illustrate the described mechanisms and serve as a starting point for further development, while empirical evaluation remains a subject for future research.

Keywords: internet of things; model context protocol; multi-agent systems; agent orchestration; large language models

1. Introduction

The rapid advancement of digital transformation has significantly increased the adoption of interconnected devices and intelligent systems across various application domains. Organizations are increasingly relying on the Internet of Things (IoT) to collect, exchange, and process data in real time, enabling more efficient decision-making, automation, and optimization of business and operational processes. As IoT ecosystems continue to grow in scale and complexity, there is an increasing need for reliable architectures capable of coordinating communication, data exchange, and intelligent decision-making across heterogeneous devices and systems.

Industry 4.0 requires the integration of people, data, services, and physical objects through the interaction of multiple technological components. Internet of Things (IoT), together with cyber-physical systems, big data, and cloud solutions, is considered one of the key enabling technologies supporting this transformation [1]. Digital farming relies on sensors and IoT devices to collect and analyse data for decision-making, enabling continuous monitoring and timely responses to ever changing conditions. Sensor networks provide real-time data, while IoT devices enable the transmission of collected information and remote access to it, supporting improved resource management, higher productivity, and reduced environmental impact [2]. Automation and the Internet of Things have become closely associated with manufacturing, logistics, and transportation, while IoT applications should be seamlessly integrated into Manufacturing Execution Systems (MES) [3].

Recent advances in Large Language Models (LLMs) and the emergence of the Model Context Protocol (MCP) provide new opportunities for standardized communication between intelligent agents, information systems, and IoT devices. These developments enable the design of more flexible and interoperable architectures for managing complex IoT ecosystems.

The objective of this research is to propose an extensible architecture for managing organizational IoT systems in which MCP serves as the central integration layer between LLM-based agents and physical IoT devices. The research is guided by the following research questions:

1. What is the current state of research and the key achievements in the fields of the Model Context Protocol, the Internet of Things, and multi-agent systems?
2. How can an extensible architecture be designed for managing an organizational system of Internet of Things agents?

The rest of the paper is organized as follows. Section 2 reviews related work on MCP, IoT and multi-agent systems while Section 3 describes the research methodology. Section 4 presents the proposed architecture and Section 5 presents the illustrative implementation of the proposed architecture. Section 6 discusses limitations and future work while Section 7 concludes the paper.

2. Literature review

This section reviews recent research related to the application of artificial intelligence (AI), Large Language Models (LLMs), the Model Context Protocol (MCP), and multi-agent systems in Internet of Things (IoT) environments. The aim of this literature review is to identify existing approaches, architectures, and research directions, with particular emphasis on the integration of MCP with IoT systems and intelligent agents. By analysing the objectives, methodologies, and main contributions of recent studies, this review provides an overview of the current state of the art, highlights existing research trends, and identifies research gaps that motivate the software–hardware architecture proposed in this paper.

Rahman et al. [4] propose a secure multi-agentic Intelligent Internet of Medical Things (IIoMT) framework for multimodal dyslexia screening by integrating specialized AI agents with Agent-to-Agent (A2A) communication and MCP-based tool orchestration. The framework combines four complementary screening modalities and enables coordinated analysis, persistent contextual intelligence across sessions, and secure inter-agent collaboration. The novelty of the research lies in integrating A2A collaboration, MCP orchestration, multimodal reasoning, and security-by-design within a single IIoMT framework.

Babun et al. [5] propose an AI-supported agent-based security model for the Internet of Things with the aim of improving the adaptability, scalability, and effectiveness of IoT security solutions. The proposed framework combines agent-based modelling, digital twins, synthetic data generation, and multi-model AI to support multiple IoT security research areas, including intrusion detection, traffic analysis, access control, software security, and taint analysis. The key contribution of the research is integrating agent-based modelling with AI-driven security analysis, enabling the generation of diverse synthetic IoT datasets for training AI models capable of detecting a broader range of cyber threats.

Li et al. [6] investigate the integration of Zero-Trust Foundation Models (ZTFMs) into IoT systems with the aim of establishing a secure and trustworthy foundation for AI-driven IoT environments. The main contribution of the research lies in presenting the first structured synthesis of ZTFMs for IoT by formalizing four core zero-trust principles and proposing a unified architecture that combines federated learning, blockchain-based identity management, micro-segmentation, and trusted execution environments to support secure, privacy-preserving, and collaborative AI in distributed IoT systems. The study also identifies current limitations and outlines future research challenges related to scalable trust management, secure orchestration, and resilient AI-driven IoT infrastructures.

Othman et al. [7] propose a 6G-assisted Internet of Things security mechanism using artificial intelligence (6G-ISM-AI) with the aim of improving the security, energy efficiency, and communication performance of AI-enabled IoT environments. The research is based on the design of a two-module architecture comprising an AI-based security module using a refined deep neural network and a power-efficient real-time 6G communication framework. The proposed approach is evaluated through simulations using the IoT Security Stack Overflow dataset and compared with existing 5G-based solutions.

He [8] proposes an artificial intelligence-based Internet of Things detection system for English translation with the aim of improving translation detection accuracy and sentiment recognition in IoT environments. The proposed approach uses deep learning techniques and implements a three-module architecture consisting of data processing, speech

recognition, and model training. The research methodology is based on the design and experimental evaluation of the proposed system.

Zeng et al. [9] investigate the application of Large Language Models (LLMs) for improving Internet of Things security by proposing an LLM-driven IoT security assistant enhanced with an IoT Chain-of-Thought (ICoT) reasoning process. The research follows a design-and-evaluation methodology, in which the proposed ICoT framework is integrated into an LLM-based security assistant and experimentally evaluated using IoT vulnerability and exploit datasets across four state-of-the-art LLMs. The contribution of the research lies in introducing the ICoT reasoning framework, which enables context-aware vulnerability analysis and personalized security recommendations.

IoT-MCP, a framework that bridges Large Language Models and Internet of Things systems through the Model Context Protocol, with the aim of enabling standardized, scalable, and reliable interaction between LLMs and heterogeneous IoT devices is presented by Yang et al. [10] and it presents IoT-MCP, a framework that follows a design-and-evaluation methodology by developing a decoupled architecture consisting of a Local Host, a Datapool and Connection Server, and IoT Devices, together with IoT-MCP Bench, a benchmark containing 1,254 IoT tasks for systematic performance evaluation. Experimental results demonstrate a 100% tool execution success rate, an average response time of 205 ms, a peak memory footprint of 74 KB, and a 99% success rate on complex prompt robustness tasks.

Surakupt et al. [11] investigate the integration of Internet of Things sensor networks with Large Language Models through the Model Context Protocol to support real-time indoor environmental prediction. The authors propose a multi-sensor architecture in which MCP orchestrates data collection, aggregation, prompt engineering, AI-based prediction, and visualization of environmental conditions. The contribution of the research is demonstration of the practical use of MCP as a standardized orchestration mechanism for coordinating data flow between multiple IoT sensors and an LLM, enabling interpretable environmental predictions from distributed sensor data.

Athanasopoulou et al. [12] investigate how Large Language Models can be integrated with Internet of Things data spaces through the Model Context Protocol to enable secure and natural-language interaction with contextual IoT data. The authors propose an architecture in which MCP serves as an intermediary between LLMs and NGSI-LD-based data spaces, allowing users to retrieve and reason over IoT data without requiring knowledge of underlying APIs or data models. The research follows a design-and-evaluation approach by developing an MCP server, implementing the solution using a FIWARE Context Broker, FastMCP, OpenAI models, and Keycloak, and experimentally evaluating its performance with multiple LLMs and a controlled set of natural-language queries. The main contribution of the study is the development of a secure, interoperable MCP-based framework that enables natural-language access to IoT data spaces while incorporating fine-grained authorization mechanisms and safeguards against LLM hallucinations.

Overall, the reviewed studies demonstrate the growing interest in integrating artificial intelligence, Large Language Models (LLMs), and the Model Context Protocol (MCP) into Internet of Things (IoT) environments [4], [9], [10], [11], [12]. Existing research primarily focuses on applying MCP as a standardized mechanism for connecting LLMs with heterogeneous IoT devices and data sources, enabling natural-language interaction, secure tool orchestration, intelligent reasoning, and real-time decision support [10], [11], [12]. Several studies also highlight the benefits of combining MCP with multi-agent architectures to improve collaboration, interoperability, and context-aware analysis in domain-specific applications [4]. Although these studies confirm the potential of MCP for facilitating communication between AI models and IoT ecosystems, they mainly address specific application domains and individual use cases, such as healthcare, IoT security, environmental monitoring, and IoT data spaces [4], [9], [10], [11], [12]. A comprehensive software–hardware architecture that employs MCP as a central integration layer for managing organizational IoT systems and coordinating intelligent agents has received limited attention, providing the motivation for the research presented in this paper.

3. Methods of the research

The research methodology is based on a literature review of existing studies related to the Internet of Things, the Model Context Protocol, Large Language Models, and multi-agent systems. Relevant scientific databases were searched to identify recent and relevant publications addressing these research areas. The selected studies were analysed with respect to their research objectives, proposed architectures, methodologies, and main contributions in order to identify current research trends and existing approaches. The analysis revealed that, despite the growing interest in MCP-based IoT integration, the number of studies addressing this topic remains limited. Furthermore, research that combines MCP with multi-agent systems in the context of organizational IoT management is still scarce, indicating that this research area has not yet been sufficiently explored.

To structure the overall research process, this paper follows the design science research methodology proposed by Peffers et al. [13], which defines problem identification; objective definition; design and development; demonstration; evaluation and communication as the core activities of design-oriented information systems research. The literature review presented in Section 2 corresponds to the problem identification and objective-definition, establishing the research gap and the two research questions addressed in this paper. The design and development activity, which constitutes the paper's main contribution that is further carried out in Section 4, where the requirements identified in Section 2 are standardized device integration, fault-tolerant connectivity and multi-agent task coordination that are translated into the proposed architecture, extending the reference design of Yang et al. with an agent-orchestration layer [10]. Consistent with the exploratory scope of this paper, the demonstration activity is carried out in Section 5 through a minimal illustrative implementation covering two scenarios, one with three concurrent Agent sessions and one combining an Agent

session with a deterministic monitor. Empirical evaluation of the deployed system is identified as future work in Section 6.

The architecture was derived in three steps. Firstly, three core requirements were derived from the research gap outlined in Section 2: standardized device integration as the first requirement, fault-tolerant connectivity as the second requirement and multi-agent task coordination as the third requirement. Secondly, the decoupled IoT-MCP paradigm from [10] was selected as the baseline architecture, given that its Connection Server fulfils the second requirement and its reliance on MCP partially satisfies the first requirement. Thirdly, this framework that was used as reference was extended with targeted components to bridge the remaining gaps: a registry-backed generic tool scheme completing the first requirement and an Agent Orchestrator resolving the third requirement. Each extension was justified by its implications for extensibility, coordination and if applicable, contrasted with the corresponding design choices in the reviewed work. Aligned with the exploratory nature of this research, evaluation is confined to a descriptive evaluation via informed reasoning and illustrative scenarios [14], [15].

4. A proposed architecture for managing organizational IoT systems

This section addresses the second research question by presenting the proposed architecture to manage organizational IoT systems through the Model Context Protocol. The architecture, its components and the workflow and scalability considerations resulting from it are discussed across the following subsections.

4.1. Architecture overview

Before describing the architecture in detail, it is useful to briefly outline the Model Context Protocol itself and the rationale for adopting it as a system's integration layer. MCP is an open, client-server protocol built on JSON-RPC that standardizes how an LLM-based application discovers and invokes external capabilities exposed by an MCP Server as tools without requiring the LLM provider or the exposed capability to share any proprietary interface. This choice was made for three reasons relevant to the organizational IoT setting addressed in this paper. First, MCP is vendor-agnostic: the same tool definitions can be consumed by any LLM from any locally or cloud hosted provider, although tool-calling success varies across models [10]. Second, MCP separates a tool's schema, which the LLM uses for reasoning, from its implementation, which remains under the MCP Server's control and execution; this clear separation is precisely what allows new devices to be onboarded, as it will be further discussed in Section 4.3, without modifying the reasoning layer above it. Third, MCP's persistent, session-based client-server connection better aligns with the continuous monitoring and control requirements of IoT infrastructure than a stateless, per-request API call would, and its adoption in recent IoT-oriented work [10], [11], [12] indicates it is becoming an accepted standard for this class of problem rather than a bespoke solution specific to this paper. To avoid ambiguity in the discussion that follows, Table 1 defines the terminology used throughout the remainder of this paper. Several of these terms are used inconsistently across colloquial usage – "Server" for instance, may refer to the MCP Server, the Connection Server, or a generic backend service that is outside the scope of this paper. This paper adopts the fixed definitions below for internal consistency; no term listed here is used elsewhere in this paper to denote a different concept.

Term	Definition
LLM	External, stateless inference service reached via API
Agent (session)	Stateful wrapper (prompt, scoped tools, history) through which the LLM is repeatedly called; not a separate model instance
Agent Orchestrator	Host application that creates, coordinates, and aggregates results across one or more Agent sessions
MCP Client	Protocol component embedded in the host; maintains a persistent 1:1 session with one MCP Server
Tools (MCP Server)	Full set of tool definitions it exposes; receives each call and dispatches it to the corresponding Tool
Tool	The specific implementation to which the MCP Server dispatches (instantiates) one active call
Device registry	Lookup structure on the Connection Server mapping a device identifier to its connection parameters
Connection Server	Resolves device identifiers and mediates persistent connections to MCUs; hosts the device registry
MCU / Sensor	The physical (simulated in the illustrated implementation) hardware layer

Table 1. Terminology used in this paper.

Mechanically, MCP operates through a 1:1 session between a client and a server. The client is embedded in the host application, here the Agent Orchestrator, through the creation of a new agent establishes a persistent connection to a given

MCP server on first contact and reuses it for all subsequent exchanges. Over this connection, the client first performs discovery of tools, known as MCP discovery, retrieving the full schema of every capability the server exposes, without executing any of them; the LLM then names a specific tool and its corresponding arguments, the MCP Client issues the corresponding invocation and the MCP server dispatches it to the corresponding Tool and returns a structured result [12]. Fig. 1 presents the proposed architecture in full, together with a legend that explains the colour coding of its components and the three interaction types. Structurally, the architecture is organized around three cooperating domains. These are an Agent domain (blue), an MCP integration domain (amber) – which intersects with the Agent domain and a physical IoT domain (pink). All of this is interconnected with external services like an LLM or a Connection Server, which sits as an intermediary between the agent and IoT Devices. The two colours of arrows denote the traffic between these domains: red for requests and blue for the data returned in response; a third, black dashed arrow denotes “instantiation”, a session-bound reference. This distinction is deliberate, not every relation in the architecture is a network call, and conflating the two would misrepresent the intended purpose. This figure is based on [10] and is extended by the addition of an Agent Orchestrator.

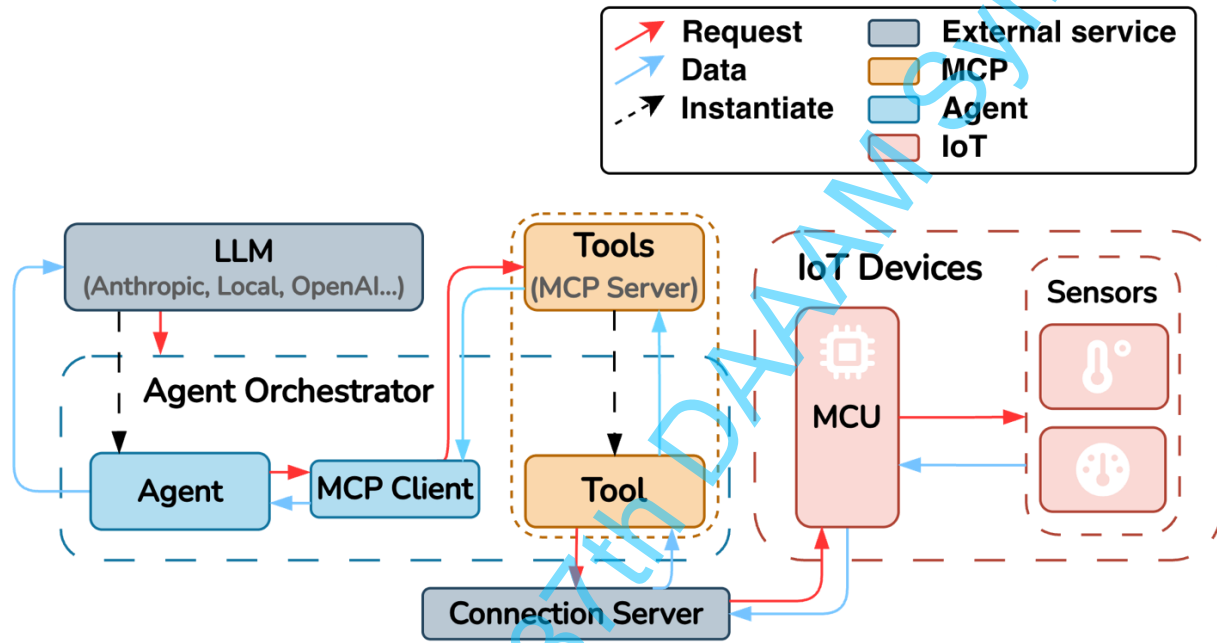


Fig. 1. Proposed architecture for agentic management of IoT devices using function-level tools exposed by an MCP Server.

The LLM, which is external in the sense of being a separately hosted inference service reached via API rather than in the sense of reasoning independently of the Agent session, carries out the only interpretive step in the entire pipeline: given the user’s natural-language request together with the tool schemas obtained during MCP discovery, it produces either a natural-language response or a structured tool call (tool name and arguments). This reasoning step is invoked from within an Agent session, held and managed by the Agent Orchestrator. The session itself is not a separate model instance, but a lightweight, stateful wrapper comprising a system prompt, a scoped list of tool schemas and the conversation history, through which the underlying LLM is repeatedly called. As noted above, the Agent Orchestrator additionally instantiates one MCP Client per connected MCP Server for each Agent session (black dashed arrow in Fig. 1) and reuses that same persistent 1:1 connection for all subsequent tool discovery and invocations, rather than establishing a new connection for every request. This then trades connection overhead at high agent counts for fault isolation and clear per-agent auditability.

Fig. 1 depicts a single Agent and a single MCU for clarity, but neither the Agent Orchestrator nor the IoT Devices domain is limited to one instance. The Agent Orchestrator may hold multiple concurrent Agent sessions, one per task, device or organizational domain identified during task decomposition. In the same way the IoT Devices domain may comprise multiple MCUs each with one or more sensors or devices attached to it [10]. The architecture’s single-instance components are shared across all active Agent sessions and all connected MCUs alike, which is precisely what allows the number of agents and the number of devices to scale independently of one another, as discussed further in Section 4.3.

The MCP domain is deliberately split into two components rather than being represented as a single component in order to make an important architectural distinction explicit: Tools (MCP Server) denotes the server together with the full set of tool definitions it exposes, discoverable in full at any point in time, whereas a Tool denotes the specific implementation that serves an active call. Tool selection is performed by the LLM, which names a tool and its arguments on the basis of the schemas obtained during discovery, resolution is performed by the MCP Server which receives the call over the session maintained by the MCP Client and dispatches it to the implementation that is registered under that name.

Traffic between the Agent and MCP domains therefore terminates on the server, as shown in Fig. 1, while the black dashed arrow between the two components denotes the internal dispatch relation rather than a network call labelled as “Instantiation” in Fig. 1. The dashed boundary drawn around Tool and Tools is not itself a component, it carries no traffic and serves only to indicate that a given Tool belongs to that MCP server (Tools), which matters once an organization operates more than one MCP server. In this case, Fig. 1 operates over just one MCP Server. The central design intent behind this arrangement is modularity: tools in the proposed architecture are defined generically per function: `read_sensor(sensor_id, sensor_type)` or `start_machine(machine_id)` tool, rather than a per physical device, with the mapping to a specific sensor or machine resolved through a device registry maintained by the Connection Server. This differs from the design adopted by Yang et al. [10], in which each MCP server is built to control a specific sensor function, so that adding hardware supporting a new function requires defining and deploying an additional, purpose-built MCP Server; it also differs from the multi-sensor MCP orchestration described in [11], which coordinates several sensors but remains bound to a fixed predetermined set rather than a dynamically extensible registry. Athanasopoulou et al. likewise exposes generic tools from a single MCP Server, parameterized by entity type and identifier, but over a read oriented data space. The present design applies the same principle to live device connections [12]. Under the proposed generic tool design, a new machine or sensor is onboarded by adding an entry to the device registry.

The device registry is a lookup structure maintained on the Connection Server, not a Tool: it associates a device identifier with its physical connection parameters (address, protocols, attached MCU or sensors). These parameters are never exposed through MCP discovery, device identifiers may be exposed through a dedicated tool, but the physical mapping remains hidden. A Tool’s implementation passes the device identifier to the Connection Server, which resolves it against the registry and forwards the request to the corresponding MCU. The Tool itself therefore never handles physical connection parameters, and a call crosses the boundary between the MCP and IoT domains once rather than twice. This placement separates two kinds of organizational growth: onboarding an additional device of a function already covered by an existing tool (e.g. another temperature sensor) requires only a new registry entry, leaving the tool’s schema and LLM-visible capabilities unchanged, whereas a genuinely new function requires a new tool and schema deployed on the MCP server.

In addition to hosting the device registry described above, a responsibility introduced by the generic tool design and absent from [10]’s device-specific server model, a dedicated Connection Server mediates all traffic to and from the microcontroller units (MCUs) and their attached sensors and devices. This layer’s core connection-management role is retained from the decoupled architecture proposed in [10] essentially unchanged, since maintaining persistent device connections, buffering requests and absorbing disconnections without stalling the reasoning layer above it was adopted as an established element of the reference design [10]; the contribution of the present architecture lies in the registry it now additionally hosts, together with the Agent Orchestrator and generic tool design above it rather than in re-engineering the connection-handling mechanisms itself. Sensor readings and command acknowledgments follow the same path in reverse, propagating from the MCU through the Connection Server, the Tool, the MCP Server and the MCP Client, back to the Agent session and ultimately the LLM, which reasons over the returned value to produce a final response or a new subsequent action.

4.2. System workflow

While Section 4.1 described the architecture’s static structure, this section traces a single representative task through it, showing the components and interactions defined above operating in sequence – in the same vein as the request/data flow trace shown by Yang et al. [10]. The example used is a natural-language request to read a specific temperature sensor, invoking the `read_sensor(sensor_id, sensor_type)` tool introduced in Section 4.1; this task is chosen because it exercises every component in Fig. 1, in both request and data form.

The resulting sequence proceeds as follows:

1. The user’s natural-language request is received by an Agent session, held by the Agent Orchestrator. The input string would be “What is the current temperature reading for sensor TEMP-04?”
2. The Agent gets paired with an MCP Client, which creates a 1:1 connection to the MCP Server.
3. The MCP Server sends its schemas to the MCP Client.
4. The LLM, invoked from this session, interprets the request and produces a structured tool call: `read_sensor(sensor_id="TEMP-04", sensor_type="temperature")`.
5. The MCP Client forwards this call to the MCP Server over the existing persistent connection, without repeating discovery.
6. The MCP Server dispatches the call (instantiates) to the corresponding Tool implementation.
7. The Tool forwards the request, still identified by “TEMP-04”, to the Connection Server, which resolves the identifier against the device registry and relays the request to the targeted MCU within the IoT system.
8. The MCU reads the attached sensor and returns the raw value to the Connection Server.
9. The value propagates back through the Tool and the MCP Server to the MCP Client and from there to the Agent session.
10. The LLM reasons over the returned value and produces the final natural-language response to the user.

Each step in this sequence corresponds directly to a labelled component. No step introduces behaviour beyond what was already defined there. This correspondence is deliberate: the workflow is intended to show that the architecture's components compose into a complete request-response cycle, rather than to introduce new mechanisms. The sequence assumes a single Agent session is sufficient for the task. When a request would span multiple devices, machines, sensors or organizational domains, the Agent Orchestrator performs a decomposition step before this sequence begins, determining how many Agent sessions the task requires and assigning each a scoped subset of the request, thus mirroring the multi-sensor coordination role demonstrated in [11]. Each resulting Agent session then independently follows the same sequence described above against its own MCP Client.

4.3. Scalability

The architecture scales along two independent axes: the number of devices under management and the number of agents required to handle a given task. Device-count scaling follows directly from the generic, registry-backed tool design established in Section 4.1, adding another instance of an already supported device function requires only a new device registry entry, leaving the MCP Server's function and capability unaltered. Growth in the number of distinct device functions still requires a new tool, but this is assumed here to occur less often than the addition of further devices of already known types in a typical organizational deployment. This does not eliminate schema growth entirely, however: an organization operating a very large number of distinct device functions would still accumulate a correspondingly large tool set and a sufficiently large number of simultaneously discoverable tools is known to degrade LLM tool-calling accuracy, independent of any property specific to this architecture [16].

Agent-count scaling follows from the decomposition step in Section 4.2, rather than a fixed number of agents, the Agent Orchestrator determines how many Agent sessions a given task requires at request time, allowing the same orchestrator to serve requests of varying scope without reconfiguration. Because each Agent session instantiates its own MCP Client per MCP Server, this flexibility then comes at the cost of connection overhead growing with agent count rather than task count. Multiple concurrent sessions are coordinated centrally, not peer-to-peer, each reports its result to the Agent Orchestrator, which aggregates them and flags conflicts for resolution rather than letting sessions act on one another directly. This corresponds to an orchestrator-worker coordination model and avoids the ambiguity a decentralized model would introduce. Because device-count and agent-count growth are handled separately, the two scale independently of one another.

5. Illustrative implementation

In keeping with the demonstration of the design science research methodology from Section 3, this section demonstrates a minimal implementation of the proposed architecture. The scope of the implementation is intentionally downsized and restricted: it is run entirely on one computer and uses the Claude's Sonnet 4.5 model from the Anthropic API as the LLM for reasoning, it also replaces the microcontroller (MCU) layer for a simulated array of sensors. The purpose of this is to demonstrate how a simple and small code-base can be instantiated and to illustrate how the Agent Orchestrator, the MCP layer and the device registry interact during representative tasks. This does not evaluate performance, robustness or behaviour under real device conditions. It should therefore be read as a pointer to the architecture described in Section 4 and used as a starting point for more complete implementations. The two scenarios show this behaviour: a three-agent scenario covering concurrent execution and the flagging of inconsistent readings and a centralized cancellation. The implementation in total consists of four Python files, each corresponding to a distinct component of the architecture described in Section 4.1, as shown below in Table 2.

File	Fig. 1 component	Role
connection_server.py	Connection Server	Holds the device registry, resolves device identifiers and routes requests to the corresponding mock MCU.
mock_hardware.py	IoT Devices	Simulates the physical layer (MockMCU, MockSensor, MockThermostat), no real hardware is used.
mcp_server.py	MCP Server / Tool	Exposes four tools via FastMCP, following the approach of Athanasopoulou et al. [12]; tool results contain no physical connection parameters.
agent_client.py	Agent Orchestrator / MCP Client	Performs MCP discovery, gives each Agent session its own MCP Client over stdio transport, and implements the Agent Orchestrator: registry-driven decomposition and deterministic aggregation in Scenario 1, and LLM-selected role decomposition and cancellation authority in Scenario 2.

Table 2. Mapping of illustrative components to Fig. 1.

For simplicity, the Connection Server is implemented as a module that is loaded within the MCP Server process and each Agent session starts its own MCP Server subprocess over stdio transport, meaning that the simulated thermostat state is shared between processes through a file. This departs from the single shared MCP Server of Fig. 1, which would require a network transport.

The *mcp_server.py* in this case exposes four tools, each operating over the device registry rather than being bound to an individual physical device. `read_sensor(sensor_id, sensor_type)` returns the current reading for a specified sensor; `list_sensors()` returns the full set of currently registered device identifiers, allowing the Agent Orchestrator to decompose tasks over registered devices without prior knowledge of the deployment; `set_thermostat(room_id, target_temp)` issues a control command to a specified thermostat; and `read_thermostat(room_id)` returns a thermostat's current temperature and target setpoint. Together, these tools cover both sensing (`read_sensor`, `read_thermostat` and `list_sensors`) and actuation with `set_thermostat`, allowing the scenarios below to exercise both the request, response and control pathways through the architecture.

Two scenarios were executed against this implementation. In Scenario 1, the request "Check all sensors. Normal ranges: temperature 18–24 °C, humidity 30–60 %. Report which sensors, if any, are outside their normal range" was partitioned by the Agent Orchestrator into individual sub-tasks per registered sensor, a deterministic division across the device registry instead of an LLM-driven process. Three concurrent Agent sessions were executed, each scoped strictly to a single sensor alongside the `read_sensor` tool, with each connected via a dedicated MCP Client. Rather than employing the LLM for aggregation, their outcomes were combined by deterministic logic: every reading was evaluated against its specified target range, and measurements from sensors of identical type exhibiting discrepancies greater than 2 °C were flagged for manual review rather than resolved silently. Sensor readings are sampled at random from ranges that straddle the stated normal ranges, so that out-of-range and inconsistent readings are likely to occur.

In Scenario 2, the request "Set the office room temperature to 28 °C" was provided to the LLM, which identified and parameterized two distinct roles from a predefined set: a controller, instantiated as an Agent session restricted solely to `set_thermostat`, and a monitor, realized as a non-LLM deterministic polling loop that repeatedly queried `read_thermostat` through its own MCP Client until the setpoint was achieved. The monitor observed the target value directly through the simulated device state modified by the controller, without maintaining any direct reference to or receiving messages from the controller session. Upon completion of the monitoring task, the Agent Orchestrator assumes responsibility for terminating the controller session if it remains active; during the reported trial, the controller session had already exited after dispatching its solitary command. Session lifecycle management, specifically instantiation and termination, remained strictly under the centralized authority of the orchestrator throughout, with no session storing references to or invoking methods on another, thereby illustrating the centralized orchestration architecture described in Section 4.3.

Scenario	Decomposition	Sessions	Observed behaviour
1. Environmental check	Registry-driven (one sub-task per sensor: TEMP-04, TEMP-05, HUM-01)	3, concurrent	TEMP-04 16.3 °C, TEMP-05 26.2 °C and HUM-01 60.3 % outside range; TEMP-04/TEMP-05 flagged as inconsistent (9.9 °C apart) rather than silently resolved
2. Thermostat control	LLM-selected roles from a predefined set (controller, monitor)	1 Agent session & 1 deterministic monitor, concurrent	Simulated thermostat reached the 28 °C target; monitor signalled completion; controller had already completed after its single command, so no cancellation was required

Table 3. Observed behaviour in one representative run of each illustrative scenario.

The implementation thereby shows that the architecture can be realized with minimal effort and illustrates how its coordination mechanisms are designed to function. It asserts no claims regarding performance, resilience or behaviour at the device scales, agent count and or concurrency levels that is detailed in Section 4.3, which represent avenues for the evaluation outlined in Section 6. The observed behaviour of the illustrative scenarios was also not repeated across multiple runs to establish statistical reliability of the conflict-detection or cancellation logic; both remain directions for future empirical evaluation. The source code of the illustrative implementation, together with the output of the reported run is publicly available on GitHub [17].

6. Discussion and Future Work

The architecture, workflow and scalability analysis in Section 4, together with the illustrative implementation in Section 5, address the second research question by proposing how an extensible, organizationally scalable IoT management system can be built with MCP as its central integration layer, and by illustrating its core mechanisms through a minimal implementation. Several limitations of the current work point directly to future research directions rather than undermining this central contribution.

The illustrative implementation is deliberately minimal in its nature. It runs locally on a single machine with simulated devices and hardware and was executed purely to demonstrate the behaviour rather than to quantify it. A natural next step is a full empirical evaluation, following a methodology comparable to the IoT-MCP Benchmark introduced by Yang et al. [10], extended to explicitly benchmark multi-agent decomposition and coordination rather than single-agent tool execution alone, together with deployment against real MCUs and sensors rather than simulated hardware. Such an evaluation should also compare single-agent and multi-agent execution of the same tasks, since only this comparison can then establish whether the cost of decomposition is justified in real-world organizational applications.

Two further limitations concern the architecture's behaviour at scale, both identified analytically in Section 4.3 rather than tested empirically. Firstly, even though the generic tool design keeps the tool set proportional to the number of distinct device functions rather than the number of devices, a sufficiently large organizational tool set is still expected to degrade LLM tool-calling accuracy [16]. In [16] the accuracy dropped by 7-85%, mitigating this would require a tool-routing or retrieval mechanism that exposes only a task-relevant subset of tools to a given Agent session, rather than the full organizational catalogue. Secondly, the Agent Orchestrator's role as the sole point of task decomposition and result aggregation makes it a natural candidate for a bottleneck under high agent concurrency, distributing or parallelizing this role without compromising the centralized-authority model established in Section 4.3 is left for future work.

The device registry, while removing the need for low-level or code-level changes when onboarding new hardware, still depends on its entries being accurate; maintaining this accuracy at organizational scale is an operational responsibility not addressed by the architecture itself. Automated mechanisms that populate and validate registry entries directly, rather than relying on manual maintenance would strengthen this aspect of the design. A comparable limitation is acknowledged in [12], where the metadata describing the available entity types is created manually and its automated generation is left as future work.

The thermostat control scenario in Section 5 also introduces actuation, not only sensing, and with it a class of concerns this paper does not address. For instance, authorization of which agent may invoke actuation-capable tools and protection against adversarial commands reaching physical infrastructure. Integrating agent-based security modelling [5] or zero-trust principles for device and identity management [6] into the MCP and Connection Server layers would be a valuable extension for any deployment involving actuators rather than sensors alone. Relatedly, any safety-critical operating threshold in such a deployment should be enforced deterministically at the Connection Server or MCU level, independent of agent or LLM reasoning, rather than relying on correct LLM behaviour at the moment it matters most. Concretely, the Connection Server can validate each actuation request against permitted setpoint ranges and a minimum interval between successive commands held in the device registry, rejecting it before the MCU is contacted, while the MCU enforces the same limits locally. A faulty or hallucinated tool parameter is then bounded by the device layer rather than by the reasoning layer. In Section 5. Implementation, only the LLM invocation logic in *agent_client.py* depends on a specific provider; the Connection Server, simulated hardware and MCP Server remain provider-agnostic, this though was not tested with a second provider.

Finally, two deployment-level directions remain open. Organizations with data-governance constraints may prefer a locally hosted LLM over a cloud-hosted one, this is supported by the architecture's vendor-agnostic design, although tool-calling success is known to vary across models: in the cross-model evaluation of [10], MCP servers achieved 99-100% success with Claude models but 77% with DeepSeek V3 and 84% with GPT-4.1 from OpenAI, a difference attributed to differing tool-calling conventions. Substituting a locally hosted model would therefore require re-validating the tool definitions against that model [10]. Separately, encapsulating organizational or procedural decision rules in a layer distinct from the MCP tools that perform execution, for instance through a dynamically loaded instruction format, could complement the present architecture without altering its core request and response mechanism, though this remains purely speculative and outside the scope of this paper.

7. Conclusion

The literature review in Section 2, which addresses the first research question, shows that existing work applies MCP mainly as a standardized connection or channel between LLMs and IoT devices within a specific application domain, while its combination with multi-agent coordination for organizational IoT management remains largely unexplored. This is the gap that motivated the architecture proposed in this paper.

In response to the second research question, Section 4 introduces an architecture in which MCP serves as a central integration layer which sits between LLM-based agents and physical devices. It extends the decoupled IoT-MCP design of Yang et al. [10] with two elements: an Agent Orchestrator that decomposes tasks into discrete Agent sessions while maintaining central control over their management and a flexible, registry-driven tool abstraction where tools are defined per function and parameterized by device identifiers resolved by the Connection Server. By using this method, expanding the device pool with devices of an existing class requires only a registry configuration change thereby leaving tools and core agent logic untouched and allowing agent and device scales to decouple from one another. Each architecture choice is presented alongside its inherent tradeoffs, such as expansion of the tool space, potential orchestrator bottlenecks and strict reliance on the accuracy of the registry.

The illustrative implementation in Section 5 shows that the architecture can be instantiated in its simplest form and at the same time, illustrate its coordination mechanisms in a registry-driven multi-sensor scenario and a role-based

control scenario. It makes no claims about performance, empirical evaluation with real hardware or comparison of single-agent and multi-agent execution. The security aspect of it remains future work.

In summary, the paper contributes an MCP-based architecture for organizational IoT management that treats multi-agent coordination as a first-class architectural concern rather than as an ad-hoc extension to a single-agent paradigm.

8. Acknowledgments

This work has been fully supported by the University of Zagreb Faculty of Organization and Informatics, institutional project Digital Twins and Smart Solutions.

9. References

- [1] Vranakova N. & Babelova Z.G. (2024). Insights into Industry 4.0 Components and Acceptance of New Technologies in Industrial Enterprises, 35th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2024, pp. 0292–0298, DOI:10.2507/35th.daaam.proceedings.040
- [2] Vrdoljak M.; Aburaia M.; Stuja K.; & Aburaia A. (2024). State of the Art Review of Smart Farming, 35th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2024, DOI: 10.2507/35th.daaam.proceedings.049
- [3] Golova J.; Mahmood K.; & Raamets T. (2021). Simulation based Performance Analysis of Production Intralogistics, Proceedings of the International Conference of Daaam Baltic Quot Industrial Engineering Quot, 2021-April, 012026, 2021, DOI: 10.1088/1757-899X/1140/1/012026
- [4] Md. Abdur Rahman; M. Minhaz Rahman; Syed Usman Jamil; M. Shamim Hossain; & Selwa Al-Hazaa (2026). A Secure Multi-Agentive Intelligent Internet of Medical Things Framework for Multimodal Dyslexia Screening, IEEE Internet of Things Magazine, 2026, pp. 1–7, DOI: 10.1109/MIOT.2026.3703027
- [5] Leonardo Babun; Luis Puche Rondon; Daniel P. Syed; & Jeffrey S. Chavis (2024). An AI-supported Agent-based Security Model for the Internet of Things, 2024 IEEE Future Networks World Forum (FNWF), 2024, DOI: 10.1109/FNWF63303.2024.11028713
- [6] Kai Li; Conggai Li; Xin Yuan; Shenghong Li; Sai Zou; Syed Sohail Ahmed; Wei Ni; Dusit Niyato; Abbas Jamalipour; Falko Dressler; & Özgür B. Akan (2025). Zero-Trust Foundation Models: A New Paradigm for Secure and Collaborative Artificial Intelligence for Internet of Things, IEEE Internet of Things Journal, Vol. 12, No. 22, 2025, pp. 46269–46293, DOI: 10.1109/JIOT.2025.3603957
- [7] Nashwan Adnan Othman; Sarah Kadhim Mohsin; Ali Ihsan Alanssari; Eay Fahidhil; Salim Yahya Kasim Khalil; & Abdul Hussein (2024). 6G-Assisted Internet of Things Security Using Artificial Intelligence, 2024 IEEE 9th International Conference on Engineering Technologies and Applied Sciences (ICETAS), 2024, DOI: 10.1109/ICETAS62372.2024.11120167
- [8] Junchen He (2024). Models of Internet of Things Detection System for Translation Based on Artificial Intelligence, 2024 First International Conference on Software, Systems and Information Technology (SSITCON), 2024, DOI: 10.1109/SSITCON62437.2024.10796564
- [9] Mingfei Zeng; Ming Xie; Xixi Zheng; Chunhai Li; Chuan Zhang; & Liehuang Zhu (2025). Large Language Model-Driven Security Assistant for Internet of Things via Chain-of-Thought, IEEE Internet of Things Journal, Vol. 12, No. 24, 2025, pp. 51832–51841, DOI: 10.1109/JIOT.2025.3577675
- [10] Yang, Ningyuan; Lyu, Guanliang; Ma, Mingchen; Lu, Yiyi; Li, Yiming; Gao, Zhihui; Ye, Hancheng; Zhang, Jianyi; Chen, Tingjun; & Chen, Yiran (2025). IoT-MCP: Bridging LLMs and IoT Systems Through Model Context Protocol, Wintech 2026 Proceedings of the 2025 ACM Workshop on Wireless Network Testbeds Experimental Evaluation and Characterization, 2025, pp. 73–80, DOI: 10.1145/3737895.3768303
- [11] Surakupt K.; Akamatsu S.; Hashimoto H.; Yuki Fujimoto; Kashihara S.; & Visoottiviseth V. (2025). Indoor Environment Prediction with Multiple Sensors and Generative AI via MCP Integration, Proceedings of the 9th International Conference on Information Technology Incit 2025, 2025, DOI: 10.1109/InCIT66780.2025.11276064
- [12] Athanasopoulou A.; Fotiou N.; & Chatzopoulos A. (2026). Interacting with IoT Data Spaces Using LLMs and the Model Context Protocol, Sensors, Vol. 26, No. 4, 1193, 2026, DOI: 10.3390/s26041193
- [13] Peffers, K.; Tuunanen, T.; Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research, Journal of Management Information Systems, Vol. 24, No. 3, pp. 45-77, ISSN 0742-1222, DOI: 10.2753/MIS0742-1222240302.
- [14] Hevner, A. R.; March, S. T.; Park, J. & Ram, S. (2004). Design science in information systems research, MIS Quarterly, Vol. 28, No. 1, pp. 75-105, DOI: 10.2307/25148625
- [15] Venable, J.; Pries-Heje, J. & Baskerville, R. (2016). FEDS: A framework for evaluation in design science research, European Journal of Information Systems, Vol. 25, No. 1, pp. 77-89, DOI: 10.1057/ejis.2014.36
- [16] Kate, K.; Pedapati, T.; Basu, K.; Rizk, Y.; Chenthamarakshan, V.; Chaudhury, S.; Agarwal, M. & Abdelaziz, I. (2025). LongFuncEval: Measuring the Effectiveness of Long Context Models for Function Calling, arXiv preprint, DOI: 10.48550/arXiv.2505.10570
- [17] Kadežabek, B. (2026). IoT-MCP-Illustrative-Example [Software], Available at <https://github.com/bkadezabek/iot-mcp-illustrative-example>, Accessed from: 2026-09-23.