VJS Pranavasri[1], Leo Francis[1], Gaurav Pal[1], Ushasri Mogadali[1], Anuradha Vattem[1], Karthik Vaidhyanathan[1], and Deepak Gangadharan[1]

[1]Smart City Research Centre, IIIT Hyderabad

April 03, 2024

Exploratory Study of oneM2M-based Interoperability Architectures for IoT: A Smart City Perspective

VJS Pranavasri*, Leo Francis*, Gaurav Pal, Ushasri Mogadali, Anuradha Vattem
Karthik Vaidhyanathan, Deepak Gangadharan
Smart City Research Centre, *IIIT Hyderabad* India
vjs.pranavasri@research.iiit.ac.in, leojfrancis.now@gmail.com,
{gaurav.pal, ushasree.mogadali,anuradha.vattem}@research.iiit.ac.in
{karthik.vaidhyanathan, deepak.g}@iiit.ac.in

***Abstract*—The advent of the Internet of Things (IoT) has ushered in transformative possibilities for smart cities, with the potential to revolutionize urban living through enhanced connectivity and data-driven decision-making. However, the effective realization of IoT in smart cities hinges upon the seamless interoperability of diverse devices and systems. To address this critical need, the oneM2M standards initiative has emerged as a foundational framework for IoT interoperability. In this research paper, we perform an exploratory analysis of three prominent open-source oneM2M based interoperability systems—Mobius, OM2M, and ACME. We leverage an existing large-scale system provided by our Smart City Living Lab deployed at IIIT Hyderabad, sprawling a 66-acre campus featuring over 370 nodes across eight verticals. We investigate the architectural characteristics of each solution, considering their strengths and limitations in facilitating IoT interoperability. Through this analysis, our paper aims to provide valuable insights for stakeholders seeking to implement IoT interoperability solutions in the context of smart cities. By evaluating the strengths and limitations of Mobius, OM2M, and ACME, we seek to offer guidance for selecting the most suitable solution. Our analysis reveals that the optimal framework choice depends on specific quality constraints: Mobius excels in performance, while ACME offers advantages in ease of setup for smaller-scale implementations.**

***Index Terms*—interoperability, software architecture, smartcity, data platform architecture, oneM2M, IoT, performance analysis**

I. Introduction

The Internet of Things (IoT) is a driving force of technological innovation, transforming industries through real-time data collection, exchange, and processing. For the full potential of IoT to be realized, interoperability is crucial. It ensures seamless communication between diverse devices and systems, underpinning efficient data exchange and integration. Among multiple interoperabilty frameworks, such as AWS IoT Core [1] and Azure IoT [2], oneM2M[1] has emerged as a global standards initiative providing a comprehensive IoT interoperability standard. Within oneM2M standards, several interoperability architectures have emerged as potential solutions to facilitate data exchange and integration in smart cities [3]. The design and implementation of these interoperability frameworks play a crucial role in determining the overall performance and quality of services in smart city applications. The rapid growth of IoT devices within these smart urban ecosystems underscores the necessity for a robust interoperability layer. Such a layer is indispensable for facilitating the smooth operation of smart city applications by enabling diverse IoT devices and systems to interact without barriers, thereby supporting the scalability, security, and reliability of the smart city network.

Popular interoperability solutions for smart cities include Mobius [4], OM2M [5], and ACME [6]. Each of these solutions, notably open-source, differ in their technology stacks and approaches. We focus on open-source options for transparency and accessibility, enabling stakeholders to make informed decisions when selecting an interoperability solution for their smart city deployment. Real-world deployments are considered key for evaluation. We use a large-scale Smart City Living Lab at IIIT Hyderabad to test these solutions and explore their architectures thoroughly and gain practical insights.

This paper is the first to explore state-of-the-art oneM2M-based interoperability solutions. Our evaluation shows Mobius consistently outperforming OM2M and ACME. In stress tests, Mobius scales exceptionally well and maintains low latency. This study aims to guide software architects to make informed choices for robust interoperability in smart cities.

II. Motivation

Our large-scale Smart City Living Lab at IIIT Hyderabad, with over 370 nodes across eight verticals, provides a real-world testing ground for smart city solutions. We rely on OM2M by Eclipse[2] as the oneM2M interoperability layer within our deployment. As our live deployment expanded, we observed the potential for a bottleneck developing due to the growing number of nodes incorporated into the system. This potential bottleneck motivates our research into the performance of different oneM2M-based interoperability

*VJS Pranavasri and Leo Francis contributed equally to this work

[1]https://onem2m.org/technical/published-specifications/release-2

[2]https://www.eclipse.org

architectures within large-scale IoT settings. We evaluate if specific architectural choices within interoperability frameworks significantly impact performance, allowing the selection of the best-suited solution for large-scale use. By scrutinizing prominent solutions, we aim to ascertain whether any of these solutions inherently provide a significant performance boost. Such solutions, when combined with a well-designed distributed architecture, have the potential to yield even more remarkable results.

This research goes beyond performance comparisons. We aim to understand if and how fundamental architectural differences of interoperability solutions influence their ability to handle vast IoT networks. Our insights will aid smart city stakeholders, in particular architects in making informed choices for building scalable, efficient, and interoperable systems.

III. Study Design and Execution

In this section, we provide a comprehensive overview of the purpose and structure of this paper, breaking down our research into three fundamental components: the experimental design, implementation, and results. To ensure a systematic and structured approach to our investigation, we adopt the Goal Question Metric Approach (GQMA), as proposed by Basili et al. in their seminal work [7]. This approach serves as the guiding framework for our research, allowing us to define the goals precisely, formulate research questions, and establish performance metrics.

A. Goals

The primary objective of this study is to assess and compare multiple interoperability architectures based on the oneM2M standards, shedding light on their key characteristics, performance metrics, scalability, and adaptability. These objectives can be defined as follows (following the GQMA):

Analyze the architecture of interoperability systems for IoT
For the purpose of finding out their impact on performance
By performing multiple intensive experiments
From viewpoint of software architects and researchers
In context of Smart Cities

B. Research Questions

To achieve this goal, we formulate the following research questions:

Research Question 1 (RQ1): *What are the key characteristics and components of each interoperability architecture based on OneM2M standards?*

This question aims to provide an overview of the architectures under consideration, elucidating their fundamental features and functionalities.

Research Question 2 (RQ2): *How do different interoperability frameworks compare with respect to metrics such as latency, throughput and resource utilization?*

By investigating this question, we seek to evaluate the performance of these architectures across various scenarios, highlighting their strengths and limitations.

Research Question 3 (RQ3): *What are the scalability and adaptability aspects of each architecture to accommodate large-scale IoT deployments?*

This question delves into the scalability and adaptability of each architecture, addressing their ability to meet the demands of extensive IoT deployments within smart cities.

C. Experiment Design

We employed a systematic approach to conduct a thorough comparative analysis of the three interoperability systems (Mobius, OM2M, and ACME). This section outlines our methods for this analysis, elucidating the data collection process, the evaluation criteria chosen, and the tools and frameworks utilized.

To ensure a consistent testing environment and maintain uniformity across the systems, we set up all three systems using Docker. The host system featured an $x86\_64$ architecture with an Intel i5 12500 processor, 16 GB RAM, and Ubuntu 20.04. This hardware configuration choice provides a stable foundation for our comparative analysis. Additionally, the host system was connected via Ethernet to minimize network-related variations. For reference and transparency, we have made the Dockerized system specifications open source[3]. This step was crucial to precisely defining the system specifications for all three systems, ensuring that each operated within an equivalent technological context.

Three important performance metrics—*latency, concurrent users, and requests per second*—were given priority in our research. IoT system efficiency and responsiveness can be evaluated using these measures. Real-time applications require low latency, and scalability is ensured by evaluating the system's ability to support numerous simultaneous users. Additionally, monitoring the number of incoming requests per second offers crucial information about how effectively the system processes data. We seek to ensure that IoT applications and systems satisfy the requirements of various use cases and user expectations by concentrating on these parameters, ultimately maximizing their performance and user experience.

For data collection, we needed to select a load testing tool that would allow us granular control and less overhead; we looked at the performance analysis performed by Pradeep et al. [8]. We leveraged Locust [9], a comprehensive testing framework renowned for its versatility and per-user control capabilities. This tool allowed us to configure various testing patterns to assess the performance of the three systems thoroughly. Our evaluation encompassed two distinct scenarios:

1. Synthetic Workload: In this scenario, we populated each of the OneM2M systems with three verticals (Application Entities - AEs) containing 150 nodes each, simulating a high-load testing environment. We conducted three types of tests—High Density Periodic, Sporadic, and a Poisson Distribution—for data retrieval (GET) and data population (POST). We call these the pattern tests. This combination resulted in 6 different

[3]https://github.com/vjspranav/om2m-comparison

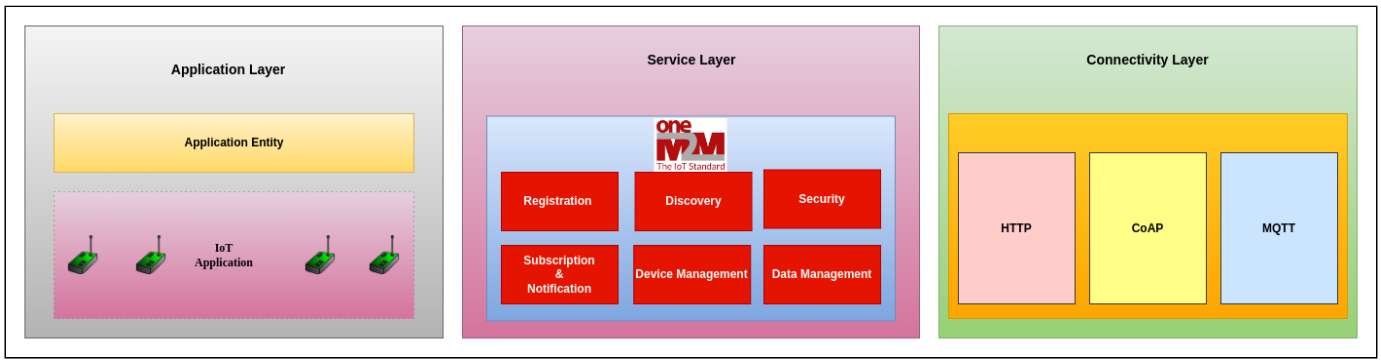

Fig. 1. OneM2M Architecture

variations of tests for each system, totalling 18 tests. Each test lasted 2 hours, and we collectively refer to these as "Pattern tests."

2. Real Workload: Here, we reconfigured the systems to mirror our production setup, comprising eight verticals and 370 nodes. Notably, real data from deployed nodes was used in these tests to replicate the conditions of a genuine production system. We conducted three types of tests in this scenario. First, the "Pattern" tests, as described earlier. Second, "Stress" tests where we incrementally increased the number of parallel users until either system crashes or performance degradation occurs, with one test each for GET and POST operations, resulting in 6 additional tests. Finally, "Emulation" tests, where we precisely simulated the data frequency generated by actual deployed nodes in each vertical over a two-hour duration. This test provided insights into how each system would perform in a production environment.

Pattern tests in each workload encompasses three distinct test scenarios designed to evaluate the system's performance under varying conditions. These scenarios include:

1) High-Density Periodic: In this scenario, a periodic surge of users simultaneously accesses the system. This test helps assess the system's response to sudden spikes in user activity.
2) Step Function: The "Step Function" scenario involves a gradual increase in the number of users, resembling a step-like pattern. This test enables us to gauge the system's sustained performance and stability as it accommodates incremental user loads.
3) Poisson: In this scenario, users post actions at random intervals ranging from 1 to 15 seconds. This test provides insights into the system's performance under unpredictable and sporadic conditions.

These pattern tests offer a comprehensive assessment of the system's responsiveness and reliability across various usage patterns and help understand its behaviour in real-world scenarios. In total, we conducted 48 comprehensive tests, all executed over Ethernet connections. In subsequent sections of this paper, we will present and analyze the results of these tests in conjunction with our examination of the architectural differences among the systems.

D. Experiment Candidates

The oneM2M architecture partitions IoT functionalities into three primary domains: the application layer, which emphasizes device-to-application connectivity; the services layer, a horizontal framework serving diverse industry applications with common IoT functions; and the connectivity layer, which is responsible for IoT device and endpoint communication. Fig 1 shows the overarching view of the layers. These layers collaborate to offer a standardized interface for application management and interaction, fostering collaborative intelligence in distributed IoT systems. All three systems we are about to discuss adhere to an architecture that aligns with the standards set forth by oneM2M. Our examination explores potential architectural distinctions beyond the variance in the employed technology stack as we strive to address the essence of **RQ1**.

1) OM2M Architecture:

The OM2M project proposes a modular architecture running

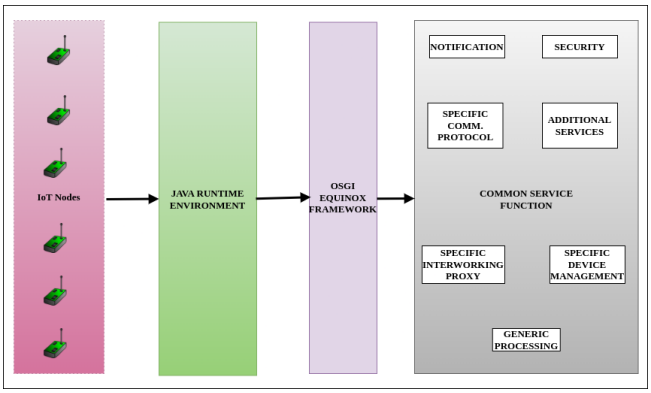

Fig. 2. OM2M Architecture

on top of an OSGi layer [10], making it highly extensible via plugins. Fig 2 shows an overview of the OM2M architecture. OM2M is a project written in Java and owned by the Eclipse Foundation. It is built as an eclipse project using maven [11] and tycho [12]. It offers a flexible Service Capability Layer (SCL) that can be deployed in an M2M network, a gateway, or a device. The SCL comprises small, tightly coupled plugins,

each offering specific functionalities. Plugins can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot. They can also detect the addition or removal of services via the service registry and adapt accordingly, facilitating the extension of the SCL. It enables the binding of multiple communication protocols, allowing interoperability with different devices and systems. The currently specified protocols are HTTP, CoAP, MQTT, and WebSocket, and the specified serialization formats are XML, JSON, and CBOR (Concise Binary Object Representation).OM2M embraces a RESTful methodology, featuring open interfaces that facilitate the development of services and applications, irrespective of the underlying network. The architecture also includes an autonomic computing service called ACS (Autonomic Computing Service) for self-configuration of M2M communications. ACS operates as an expert system, emulating human decision-making abilities to solve complex problems by reasoning about knowledge. It enables dynamic discovery and self-configuration of M2M applications based on device profiles and application roles. We use version V1.4.1, which is the latest version at the time of writing the paper.

2) ACME Architecture:

ACME is a lightweight implementation primarily for educational and experimental purposes. It leverages Flask, a micro web framework in Python, and TinyDB, a NoSQL database, to create a modular architecture that offers flexibility and extensibility, as shown in fig 3. ACME covers a wide range of oneM2M service features, including AE registration, blocking and non-blocking requests, notifications, resource addressing, semantic queries, and time synchronization. This comprehensive feature set ensures efficient communication and resource management. It supports different protocol bindings, including HTTP (TLS and CORS), MQTT (MQTT-S), and WebSocket. This versatility allows ACME to communicate with a variety of devices and systems. ACME supports various serialization types, including JSON and CBOR, enabling data exchange in different formats. ACME can run on different runtime environments, such as Generic Linux, Mac OS, MS Windows, and Jupyter Notebooks. This adaptability enhances its deployment flexibility. ACME primarily serves as an educational tool for learning IoT and M2M communication concepts, which is valuable for students, researchers, and developers. They can use ACME to experiment with IoT and M2M scenarios, test plugins, and explore resource management techniques in a controlled environment. Hence, for the very same reason, ACME comes with the following limitations:

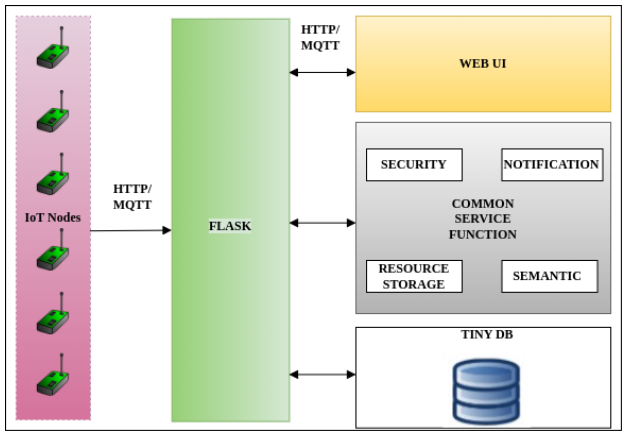

Fig. 3. ACME Architecture

1. Educational Focus: ACME is primarily designed for educational and experimental purposes, lacking production-grade requirements.

2. Security Considerations: While suitable for learning and experimentation, ACME may not incorporate advanced security features required for secure IoT applications.

3. Limited Protocol Support: ACME's protocol support is limited to what Flask offers and does not cover all communication protocols used in IoT and M2M scenarios.

ACME's architecture is designed to provide a lightweight, flexible, and extensible framework for IoT resource management and experimentation. It is best suited for educational and experimental purposes and may not meet production-grade IoT deployments' security and performance requirements.

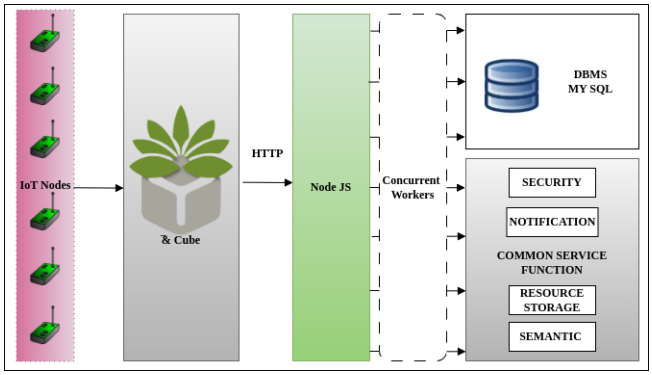

Fig. 4. Mobius Architecture

3) Mobius Architecture:

Mobius is a server software platform designed to manage and store data from IoT devices. It is developed using JavaScript language based on Node JS [13], which is a high-performance network server that supports asynchronous I/O. Mobius supports MySQL as the database and HTTP, MQTT, CoAP, and WebSocket as communication protocols. The software architecture of Mobius is based on the concept of functional components, which allows for easy customization and scalability. The components include the Mobius server, which manages the data from IoT devices, and the Mobius API server, which provides a RESTful API for accessing the data. Mobius also includes a web-based management console for configuring and monitoring the system. Something that mobius handles differently is that it spawns itself into multiple clusters, parallelly processing different requests, as seen in fig 4, which allows for a much higher concurrent performance boost. With its flexible architecture and support for multiple communication protocols, Mobius is a powerful platform for managing IoT data. One difference comes with

the introduction of &Cube [14], which acts as a middleware between nodes and interoperability; &Cube, together with Mobius, forms the oneM2M software system.

IV. Results

This section presents the results of our comprehensive evaluation comparing the three systems: OM2M, Mobius, and ACME. The evaluation encompasses various aspects of performance and efficiency, as detailed below. We applied a logarithmic scale to the graphs to facilitate a comprehensive comparison of response times that spanned a wide range. Therefore, even seemingly small differences on the graphs can represent significant variations in actual response times. All the results are available online and can be directly accessed[4].

A. Synthetic Workload

1) Response Time and Latency:
In fig 5, we present a plot comparing the response times of the three middleware platforms as we increase user load, contributing to the evaluation of latency as part of RQ2. We conducted these comparisons for both get and post requests, resulting in a total of six graphs. These visualizations show Mobius consistently showcasing the lowest response times, followed by ACME and OM2M. One small outlier to note is the worse performance of ACME in the POST request step scenario where response time crosses that of OM2M, deviating from all the other results we have. We attribute this to other variables and believe that ACME itself does not lead to an edge case behaviour in the given situation, given its performance across all the other scenarios.

2) CPU Stats:
Fig. 7 explores CPU usage trends in relation to the number of concurrent users. While our testing focused on a synthetic workload, the findings will likely generalize to real-world scenarios. Our investigation addressed resource utilization (RQ2), incorporating three distinct usage patterns with both GET and POST requests for each middleware system. Mobius consistently demonstrated superior efficiency, exhibiting lower CPU usage than ACME and OM2M across all patterns.

B. Real Workload

1) Response Time and Latency:
Similar to synthetic workload, we see Mobius consistently performing better, followed by ACME and lastly OM2M. These results can be seen in fig 5

2) Emulation of Real System:
Fig 6 illustrates the response time as a function of the user count in an emulation of real-world IoT workload conditions. We conducted post requests at a frequency equivalent to real-life nodes, using actual production deployment replicas. Notably, across different user counts, Mobius consistently demonstrated lower response times compared to both ACME and OM2M, addressing the latency aspect of RQ2.

System	Peak RPS	95th Avg Latency(ms)
OM2M	1	432000
ACME	14	95000
Mobius	568	1600

TABLE I
STRESS TESTING POST REQUESTS ON REAL WORKLOAD

System	Peak RPS	95th Avg Latency(ms)
OM2M	3	207000
ACME	43	42000
Mobius	124	12

TABLE II
STRESS TESTING GET REQUESTS ON REAL WORKLOAD

3) Stress Test:
In addition to our comprehensive evaluation of latency, throughput, and resource utilization, we conducted stress testing to assess the robustness and scalability of the three IoT middleware platforms: OM2M, ACME, and Mobius. Stress testing involved incrementally increasing the number of concurrent users to determine the maximum requests per second (RPS) each system could handle without crashing. We also measured the 95th percentile average latency (95th Avg Latency) at these peak RPS values. The results, presented in Tables I and II. In the stress testing of POST requests, Mobius demonstrated exceptional scalability, with a peak RPS of 568, representing a remarkable 56,700% increase in RPS compared to OM2M and a 3,942% increase compared to ACME. Additionally, Mobius achieved an impressively low 95th Avg Latency of 1600 ms under this extreme load, highlighting its efficiency even in high-stress situations. Similarly, in the stress testing of GET requests, Mobius achieved a peak RPS of 124, showing a significant 4,033% increase compared to OM2M and a 188% increase compared to ACME. Furthermore, Mobius achieved an exceptionally low 95th Avg Latency of 12 ms.
This extensive evaluation consistently positions Mobius as the leading middleware interoperability platform across various performance metrics, including latency and resource utilization. Its efficiency, especially in terms of response time and CPU usage, underscores Mobius as a robust and high-performing interoperable middleware solution for IoT deployments, effectively addressing the metrics associated with **RQ2** and highlighting its suitability for a wide range of IoT use cases.

V. Discussion

In this section, we will address the three research questions that guided our evaluation of IoT middleware platforms: RQ1, RQ2, and RQ3. These questions delve into the key characteristics, performance metrics, and scalability aspects of the interoperability architectures. In our comprehensive evaluation of these platforms, Mobius consistently emerged as the clear winner across all performance metrics. Its superior responsiveness, scalability, and efficiency underscore its potential as

[4]https://vjspranav.github.io/om2m-comparison/

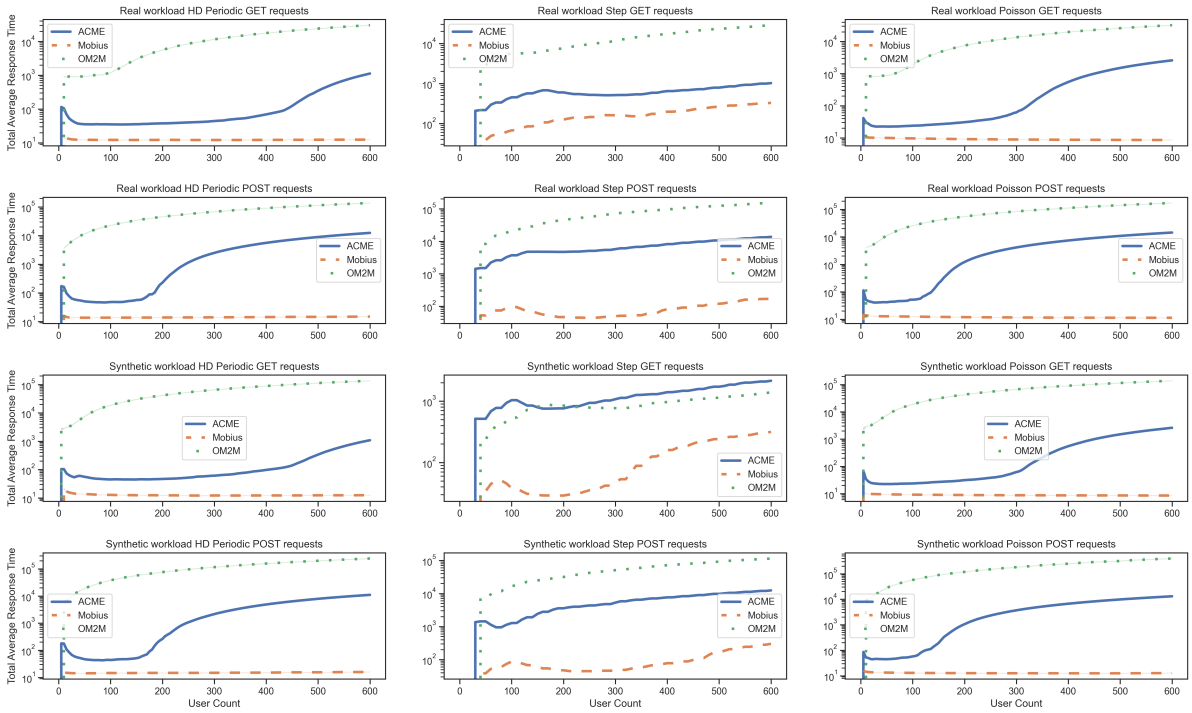

Fig. 5. Different request pattern comparison

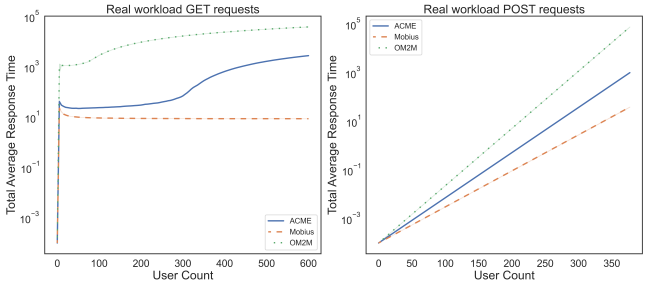

Fig. 6. Emulation of the real workload

a robust middleware solution for IoT applications. We take an in-depth look at the implementational differences between OM2M, ACME, and Mobius and how these differences may influence the software's performance.

What are the key characteristics and components of each interoperability architecture based on OneM2M standards? (RQ1)

From our study of the architectures of the systems in Section III-D, we delve into the distinctive characteristics and components of interoperability architectures based on the OneM2M standards. By examining the architectural details of these systems, we gain valuable insights into their key attributes and functionalities.

OM2M: The OM2M architecture is characterized by its exceptional modularity and extensibility, allowing the seamless integration of specialized functionalities through plugins without requiring system-wide reboots. In the context of our research, OM2M's codebase exhibits a high degree of modularity and extensibility, following good design patterns [15] and aligning with established software engineering principles. This architectural attribute aligns with the concept of a "modular monolith" [16] within the realm of Software Engineering, highlighting its structural soundness and versatility.

ACME: ACME, designed primarily for educational and experimental purposes, adopts a lightweight and modular approach. It utilizes Flask and TinyDB to offer flexibility and extensibility. The codebase itself was quite straightforward, leveraging the power Python provides.

Mobius: The Mobius architecture integrates with &Cube, enhancing its capabilities as part of a comprehensive oneM2M software system. It is worth noting, however, that the Mobius codebase exhibits a certain level of complexity and limited extensibility. The presence of substantial hardcoding within the code, while not inherently erroneous, deviates from established design patterns and best coding practices.

A key insight from this study is that well-structured, best-practice code does not always translate to optimal performance. It suggests that OM2M's architectural complexity may be excessive for specific tasks, highlighting the balance between code quality and performance.

How do different interoperability frameworks compare with respect to metrics such as latency, throughput and resource utilization? (RQ2)

Mobius exhibits a clear and substantial performance advantage within comparative interoperability framework benchmarksIV.

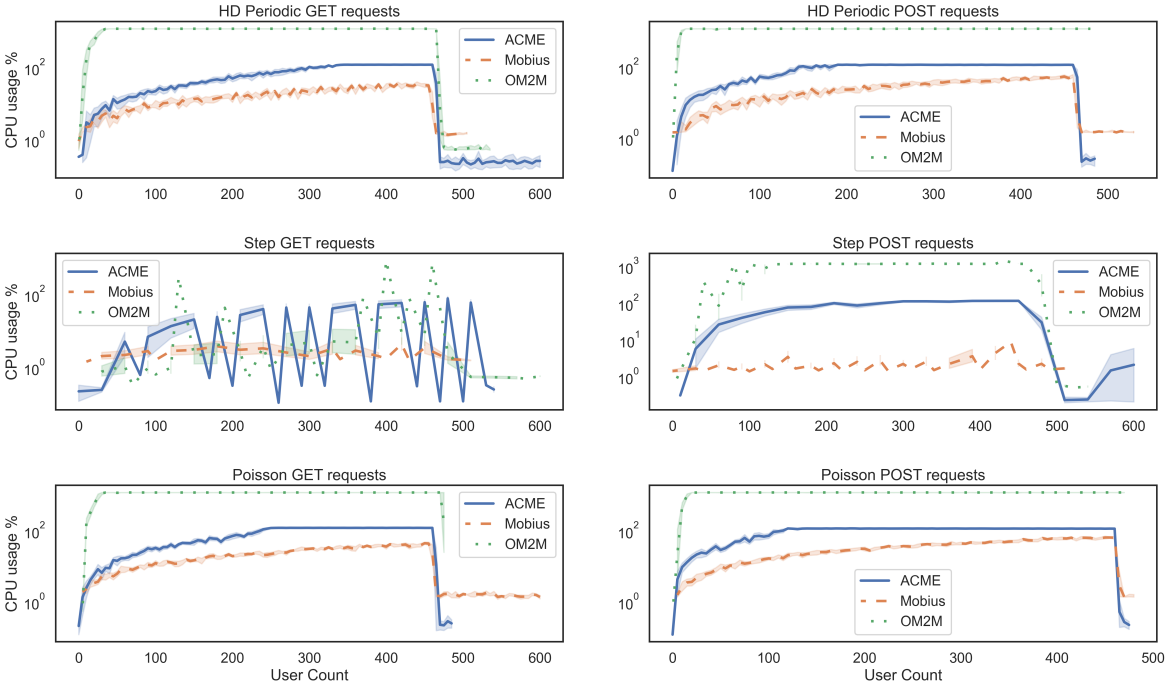

Fig. 7. CPU usage statistics with respect to number of concurrent users

In the handling of POST requests, Mobius achieves a peak rate of 568 requests per second, representing a 56,700% improvement over OM2M. This exceptional throughput, coupled with a 95th percentile average latency of 1600ms for data insertion during stress testing, demonstrates the framework's robust scalability. Mobius consistently maintains low latency for data retrieval (under 12ms), ensuring rapid information access. These empirical results underscore Mobius's leadership in terms of latency, throughput, and resource utilization. Such performance characteristics are indispensable within the context of IoT applications, where scalability and real-time responsiveness are crucial.

What are the scalability and adaptability aspects of each architecture to accommodate large-scale IoT deployments? (RQ3)

RQ3 delves into the scalability and adaptability of each interoperability framework to effectively accommodate large-scale IoT deployments. OM2M, despite its support for threading, exhibits a substantial CPU usage spike even under relatively light loads, suggesting the presence of potential bottlenecks. In contrast, ACME showcases well-structured code and leverages Python's threading capabilities, but it falls short due to incomplete adherence to the oneM2M specification. Moreover, the use of tinyDB [17] appears to contribute to its relatively better performance compared to OM2M. Mobius stands out with its highly efficient concurrency control mechanism, employing multiple replicas, precisely equivalent to the number of CPU cores, with fault tolerance. This parallel execution approach provides a substantial advantage over the other two systems, particularly in scenarios involving extensive concurrency.

A. *Threats to Validity*

While our study provides valuable insights, it is essential to acknowledge potential validity considerations. Internally, we conducted our evaluation under specific test conditions and workloads to ensure robustness. However, it is worth noting that these conditions may only partially encapsulate the richness of all real-world scenarios. Variations in workloads and use cases could yield diverse outcomes. We rigorously managed system configurations throughout our experiments to maintain consistency and minimize biases. Externally, we aimed to generalize our findings across different IoT systems, yet the chosen metrics may not cover every possible performance aspect. Additionally, our study primarily focused on performance metrics, with limited exploration of security and fault tolerance aspects. Furthermore, we conducted a fair comparison, but it is important to recognize that inherent differences in the platforms themselves may have influenced our results.

VI. Related Work

Multiple studies have contributed to the in-depth analysis of IoT architecture using system-level concepts, including resilience, scalability, interoperability, and security. One significant work by Swamy and Kota [18] describes the fundamental elements of the Internet of Things, such as its architecture, communication standards, edge computing, security issues, and real-time operating systems. It aims to stimulate research

into energy-efficient, scalable, and secure IoT applications while emphasizing interoperability and addressing new privacy concerns. In the development of IoT systems, it also identifies open research areas. By analyzing recent research advancements, categorizing them according to factors like applications and technologies, and highlighting crucial criteria and challenges for developing IoT design, our study further aligns with the system architecture of interoperable systems in IoT development, which is highlighted by Yaqoob et al. [19] as the IoT architectures. Furthermore, it offers case studies and forthcoming challenges in the subject field. The oneM2M standard, according to Alaya et al. [20] aims to improve M2M communication through the introduction of IoT-O, an ontology that enables semantic data interoperability, leading to better device interoperability and automated reasoning for autonomic behavior. The constraints of current M2M standards are addressed by this integration. A study by Mante et al. [21] suggested that a multi-layer smart city architecture that complies with the oneM2M and IUDX standards. For effective data management and secure sharing, it has layers for Data Monitoring, Storage, and Exchange. An implementation used as a proof-of-concept showed strong performance, minimal latency, and large user capacity. Our previous work [22] proposed a multi-layered distributed IoT data platform architecture for smart cities which ensures interoperability, scalability, and data transfer efficiency. While the interoperability software was replicated to enhance availability, our architecture's distributed nature still led to significant performance gains. Real-world analysis demonstrated up to 41.23% throughput improvement and 29.19% latency reduction for sensor data insertion, with over 800% throughput increase for data retrieval compared to a centralized architecture, offering significant implications for urban development and future research avenues. But there is scope for a much higher performance improvement if a better oneM2M standard system is used. In the context of IoT architecture, the oneM2M interoperability frameworks OM2M, Mobius, and ACME were the main subjects of our study.

VII. Conclusion and Future Work

In this study, we extensively evaluated three prominent IoT middleware platforms, OM2M, ACME, and Mobius, with a focus on metrics such as latency, throughput, and resource utilization. Key findings from our study highlight the importance of careful consideration in architectural design. While modular code adhering to good coding standards is essential for maintainability and codebase organization, our results demonstrate that it does not always guarantee superior performance. Proper parallelism and concurrency management emerge as crucial factors, as evidenced by Mobius consistently outperforming its counterparts.

These findings hold significant implications for IoT developers and researchers. They highlight the need to balance modularity with performance considerations in architectural design. Developers should carefully assess the scalability and concurrency handling capabilities of middleware platforms, taking into account the specific requirements of their applications. Researchers can use these insights to refine existing IoT architectures and design more efficient systems.

Future research in the field of oneM2M architectures should explore microservice-based architectures to further improve performance. By separating concerns into microservices, it becomes possible to optimize and scale individual components independently. This approach aligns with the scalability needs of IoT applications, as indicated by our findings, and presents an avenue for enhancing responsiveness and reliability.

References

[1] S. Pal, V. G. Diaz, and D.-N. Le, "Aws iot core," 2022. [Online]. Available: https://docs.aws.amazon.com/iot/

[2] Philmea, "Azure iot." [Online]. Available: https://learn.microsoft.com/en-us/azure/iot/

[3] A. Souza, J. Pereira, J. Oliveira, C. Trindade, E. Cavalcante, N. Cacho, T. Batista, and F. Lopes, "A data integration approach for smart cities: The case of natal," 09 2017, pp. 1–6.

[4] IoTKETI, 2023. [Online]. Available: http://developers.iotocean.org/archives/module/mobius

[5] M. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "Om2m: Extensible etsi-compliant m2m service platform with self-configuration capability," *Procedia Computer Science*, vol. 32, pp. 1079–1086, 12 2014.

[6] A. Kraft, "Acme," https://github.com/ankraft/ACME-oneM2M-CSE.

[7] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:13884048

[8] S. Pradeep and Y. K. Sharma, "A pragmatic evaluation of stress and performance testing technologies for web based applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, 2019, pp. 399–403.

[9] C. B. J. H. Jonatan Heyman, Lars Holmberg and H. Heyman, "Locust." [Online]. Available: https://docs.locust.io/

[10] O. W. Group, "Osgi docs home page." [Online]. Available: https://docs.osgi.org/

[11] Apache, "Maven," https://maven.apache.org/, 2023.

[12] E. Web, "Eclipse tycho," Jan 2013. [Online]. Available: https://projects.eclipse.org/projects/technology.tycho

[13] Node.js, "Node.js," 2023. [Online]. Available: https://nodejs.org/en

[14] J. Yun, I.-Y. Ahn, S. Choi, and J. Kim, "Tteo (things talk to each other): Programming smart spaces based on iot systems," *Sensors*, vol. 16, p. 467, 04 2016.

[15] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?" *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1213–1231, 2012.

[16] N. Gonçalves, D. Faustino, A. R. Silva, and M. Portela, "Monolith modularization towards microservices: Refactoring and performance trade-offs," in *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, 2021, pp. 1–8.

[17] M. Siemens, "Tinydb," https://tinydb.readthedocs.io/, 2023.

[18] S. N. Swamy and S. R. Kota, "An empirical study on system level aspects of internet of things (iot)," *IEEE Access*, vol. 8, pp. 188 082–188 134, 2020.

[19] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, 2017.

[20] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira, "Toward semantic interoperability in onem2m architecture," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 35–41, 2015.

[21] S. Mante, S. S. S. Vaddhiparthy, M. Ruthwik, D. Gangadharan, A. M. Hussain, and A. Vattem, "A multi layer data platform architecture for smart cities using onem2m and iudx," in *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, 2022, pp. 1–6.

[22] V. Pranavasri, L. Francis, U. Mogadali, G. Pal, S. S. S. Vaddhiparthy, A. Vattem, K. Vaidhyanathan, and D. Gangadharan, "Scalable and Interoperable Distributed Architecture for IoT in Smart Cities," 9 2023.