ZION: A Scalable W3C Web of Things Directory

(Article begins on next page)

22 November 2024

ZION: A Scalable W3C Web of Things Directory

Cristiano Aguzzi*, Lorenzo Gigli*†, Ivan Zyrianoff*†, Luca Roffia‡

* Department of Computer Science and Engineering, University of Bologna, Italy
† Advanced Research Center on Electronic Systems "Ercole De Castro", University of Bologna, Italy
‡ Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi", University of Bologna, Italy
§ Emails: {cristiano.aguzzi, lorenzo.gigli, ivandimitry.ribeiro, luca.roffia}@unibo.it.

Abstract—The proliferation of non-interoperable smart objects within the Internet of Things (IoT) has led to a fragmented and heterogeneous landscape. The Web of Things (WoT), particularly the W3C WoT, has emerged as a promising solution to this challenge, enabling seamless integration across IoT platforms and domains by extending known web standards. This paper introduces Zion, an open-source scalable W3C Thing Description Directory (TDD) designed to efficiently address the indexing and querying of Thing Descriptions (TDs) and the associated Web Things (WTs). Zion offers a standard API for performing CRUDL operations while supporting metadata querying through JSONPath. We further demonstrate its practical utility through real-world deployments, applying Zion to Structural Health Monitoring (SHM) and integrating it with IoT devices alongside blockchain technology. Comparative analysis with the other TDD implementations complying with the W3C standards – e.g., WoT Hive and TinyIoT – demonstrates that Zion outperforms both. It exhibits response times approximately ten times lower than those observed in the compared TDDs under high workloads.

Index Terms—Interoperability, Web of Things, Performance Evaluation.

In recent years, there has been an exponential surge in the number of smart objects connected to the Internet, giving rise to a globally interconnected ecosystem, i.e., the Internet of Things (IoT). This novel paradigm catalyzed a new generation of applications, platforms, and devices. However, the rapid development of IoT has been accompanied by a significant challenge: *the lack of interoperability*, manifesting through the proliferation of proprietary interfaces, diverse data formats, and a plethora of network protocols. Those factors lead to an unprecedented level of fragmentation and heterogeneity within the IoT landscape [1]. The lack of interoperability in IoT is a crucial challenge that hinders the expansion and evergreen adoption of IoT-based systems [2].

One of the most promising solutions to address the challenges in the current IoT scenario is the Web of Things (WoT) [3], with a particular emphasis on the W3C WoT framework [4]. The W3C WoT extends existing Web standards and technologies to provide interoperable abstractions. It achieves this by offering standardized metadata and other reusable technological components, facilitating seamless integration across diverse IoT platforms and application domains [4]. The W3C WoT has widespread as an interoperability solution across various domains characterized by heterogeneity, including but not limited to condition monitoring scenarios [5], edge caching [6], [7], Industry 4.0 [8] and mobile crowdsensing [9].

One crucial challenge within the WoT ecosystem is the efficient indexing and querying of Web Things (WTs) [10].

To address this, a novel W3C standard has been introduced, outlining the process for discovering WTs by utilizing device metadata while incorporating security and privacy features [11]. The W3C publishes the standard specification documents, but it does not provide its implementation. Instead, the engaged communities often take up the implementation task through open-source initiatives. However, there is no scalable WoT-compliant indexer, particularly concerning a domain that frequently interfaces with numerous devices, often numbering in the thousands.

To bridge this gap, we have introduced Zion, a scalable W3C Thing Description Directory (TDD). Zion represents an open-source TDD fully aligned with the W3C Discovery standard. Its core values revolve around speed, flexibility, and user-friendliness. It comprises a standard API that facilitates CRUDL (Create, Read, Update, Delete, List) operations managing TDs. The same interface supports querying TDs metadata through JSONPath, following the IETF JSONPath standard[1], and offers robust pagination capabilities.

To validate Zion's scalability, we conducted a comparative analysis with two other open-source TDD implementations, specifically, WoT Hive [12], and TinyIoT [13]. We evaluated the querying performance of each TDD while indexing various quantities of Web Things, ranging from hundreds to tens of thousands. Our results showcase that Zion service time increases linearly while TinyIoT scales exponentially. Additionally, the WoT Hive processing time was prohibiting high no matter the tested workload.

In the remainder of this paper, Section I details the W3C WoT Standards while Section II discusses other similar works in the field. Zion architecture is presented in Section III, its use cases are illustrated in Section IV, and its scalability is assessed by the performance evaluation conducted in Section V. Finally, Section VI concludes and proposes relevant future works.

I. W3C WEB OF THINGS STANDARD

Unlike traditional approaches, which often propose the creation of new protocols or middleware layers, the W3C WoT approach revolves around a descriptive information model capable of representing diverse solutions. This information model is the *Thing Description* (TD) [4].

---

[1]https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base#section-3.5.8

The TD contains detailed instructions about the nature of a device or service – i.e., a WT. In practice, the information model standardized by W3C contains a set of interaction patterns (or affordances) that a service or device is capable of supporting. Those interaction patterns have been designed after an in-depth review of the current IoT landscape and they have been found quite good to represent a vast range of use cases. The affordances group together a set of atomic functions, called operations, that clients can use to interact with the represented WT. A WT can have three types of affordances:

- **properties**: represent the inner state of a WT – e.g., the current temperature of a smart thermometer or the configuration parameters of a coffee machine. A client can perform the following operations: write read, observe, and unobserved.
- **actions**: high-level functions that WTs offer to their clients. Usually, actions operate on the physical world or modify the WT's state. Examples include: toggle a smart lamp or move a robotic arm to the desired position. A client can invoke the action and if it is a long-running function it can later query its state or cancel its execution.
- **events**: data sources that asynchronously push data to clients. Typically, alarms or expected states are modeled as events, but not regular property changes that are modeled with the Property affordances. The operations grouped under the events affordances are: subscribe and unsubscribe.

A *Protocol Binding Template* defines how clients can perform the operations described in the TD. These low-level communication requirements can be paired with security constraints thanks to the *Security Schemas*. Security Schemas are common Web authentication paradigms that can be declared to be used when invoking an action or any affordance operation. Notice that the TD only contains the methodology to access the affordance, not the sensitive information (e.g., username and password) required to interact with the WT.

In summary, the TD serves as the cornerstone of the W3C Web of Things, empowering clients to seamlessly interact with a wide range of software and hardware entities. However, the true power of WoT lies in its ability to offer the ability to dynamically discover TD at runtime. This capability ensures that clients can adapt to the ever-evolving landscape of devices and services operating across diverse protocols, whether they are known in advance or encountered on the fly. This crucial aspect of discovery is thoughtfully addressed by the normative specification of the W3C Web of Thing Discover [11], detailed in the following subsection.

A. Discovery

The W3C Web of Thing Discovery [11] focuses on the normative steps required to obtain and publish TDs over the Web. The approach to acquiring TDs adopts a two-phase architectural model, balancing the dual demands of openness and controlled access to metadata, ensuring that only authorized entities can access the necessary information.

The first phase, known as *"Introduction,"* is employed to discover one or more candidate URLs. These URLs are treated as opaque strings, deliberately devoid of any substantial metadata. During this phase, the process remains entirely open, with no restrictions applied to consumers. The candidate URLs are acquired through one of the defined introduction methods. Presently, there exist five introduction methods: well-known URLs, Direct (e.g., QR codes or manual URL provision), DNS-Based Service Discovery, CoRE Link Format and CoRE Resource Directory, and DID Documents.

Upon obtaining a set of candidate URLs, the Discoverer proceeds to the second phase, denoted as *"Exploration."* This phase encompasses the operations necessary to retrieve the TD referenced by the URLs and further processing to extract additional information. Typically, these operations are protected by security mechanisms, such as authentication tokens, ensuring that the TDs remain inaccessible to unauthorized users. Notice that the URL obtained via one introduction mechanism invariably directs to a single TD hosted by an exploration service. Discoverers must be capable of interacting with different types of exploration services:

- **Thing Description Server**: "Any web service that can be referenced by a URL and returns a TD with appropriate authentication and access controls can be used as an exploration mechanism" [11].
- **Thing Description Directory (TDD)**: serves as a WT that offers services for managing a collection of TDs describing other WTs [11]. The TDD facilitates a broader set of APIs for filtering and searching for the desired TDs.

B. Thing Description Directory

A TDD is an exploration service that can be used to retrieve and filter a list of TDs. Currently, the specification is focusing on TDD based on HTTP but, in the future, it might support other non-web-native protocols like CoAP or MQTT. Implementers of a TDD are required to support a set of compliant APIs exposed as HTTP endpoints. Currently, those APIs are grouped into three categories: things, events, and filtering. Things endpoints are further subdivided into creation, retrieval, update, deletion, and listing. Those functions represent the CRUDL operations for the set of TDs stored inside the service. The specification recommends protecting relevant resources with secure protocols and credentials. The events API allows the client to subscribe to the basic events fired by the TDD like the creation of a Thing Description, an update, or a deletion. Finally, the filtering API comprehends three different querying technologies that implementers can choose to support or not: JSONPath, XPath, and SPARQL.

II. RELATED WORK

One of the most prominent verticals of WoT research is to design and develop a mechanism to enable the W3C WoT standard to integrate with non-WoT components seamlessly. Indeed, a known shortcoming of the W3C WoT standard is the lack of out-of-the-box conversion methods to dissonant interfaces to its ecosystem. Implementation efforts are often

needed to integrate third-party Web services or other standard interfaces into the WoT. Recent advances filled that gap, providing seamless integration of RESTful Web services [14] and NGSI-based interfaces [15] to the W3C WoT ecosystem. Other efforts are on the live migration of WTs [16] to cope with the intrinsic dynamicity of IoT environments in terms of time-varying network and computational loads.

The WoT standard enables abstracting the device's physical properties and creating interoperable interfaces that facilitate seamless communication within IoT systems. However, efficient indexing and searching of WTs are fundamental aspects for the widespread adoption of WoT [3]. Numerous techniques have been proposed to address WT search challenges [17], varying in the adopted query language and overall technology. Among these, IoT-SVKSearch [18] stands out as a promising approach, supporting searches based on both spatial-temporal attributes and value-based criteria, effectively incorporating the dynamism of IoT environments into the search mechanism. GOLDIE [10] offers a hierarchical location-based WoT directory architecture that includes federated identity management and IoT-specific features like discoverability, aggregation, and geospatial queries. DBAC [19] innovates in the access-control vertical, enabling decentralized attributed access without the need for complete trust or credential provision while preserving user privacy. Other efforts have focused on indexing WTs for specific scenarios, such as indoor devices [20]. In [20], device features are automatically extracted using machine learning techniques and clustered to group similar devices. Although these works advance the state-of-the-art in WT indexing and searching, they do not align with current W3C standards for discoverability, leading to an increasingly fragmented landscape with multiple heterogeneous solutions for indexing and querying devices.

There are two other W3C-compatible implementations, namely TinyIoT [13] and WoT Hive [12]. TinyIoT holds a historical significance as the first implementation of the APIs outlined within the specification. Originating as a research project within the Fraunhofer Institute, it has since evolved into an independent open-source endeavor. The service, implemented in Go, uses an integrated LevelDB instance for the storage and querying of TDs. It supports a comprehensive feature set, including DNS-SD as an introduction service for the TDD, complete implementation of all mandatory APIs, and a JSON-Path query endpoint. While the software solution is robust, the queries are performed entirely in memory, which could potentially pose challenges when deploying TinyIoT in large-scale environments.

WoT Hive has been developed inside the European project AURORAL and wants to be the most feature-rich implementation of the Thing Description Directory APIs. The service is written in Java with the help of the Spark framework and it supports SPARQL endpoints as storage for the list of TDs. In contrast to TinyIoT, WoT Hive boasts more robust semantic and syntactic capabilities thanks to its backend support for Triple stores, supporting both JSONPath, SPARQL-based discovery and semantic validation. On the other hand, the expanded feature set compromises scalability with a large set of TDs as demonstrated in [12]. Although we utilized the WoT standard in this study, there are other efforts to solve the interoperability problem in the context of IoT [21], which are outside the scope of this work.

III. ZION ARCHITECTURAL DESIGN

Zion's software architecture is modular, ensuring scalability and maintainability. The architecture is divided into:the API Reference, Authentication, Introduction, and Persistence modules. Each of these modules serves a distinct purpose, ensuring that the software can handle the diverse requirements of an IoT device directory.

A. API Reference

The API Reference offers a comprehensive implementation of the functionalities outlined in the W3C WoT Discovery document. It is structured into three distinct sub-modules: Events, Search, and Things.

The Events module handles API endpoints related to device events, allowing the tracking and monitoring of device activities. Clients can subscribe to all events or a specific event, such as when a device is added, modified, or deleted. This real-time subscription mechanism is implemented using Server-Sent Events (SSE), ensuring immediate updates and efficient communication between the server and clients.

Within the W3C-specified Search API, advanced searching mechanisms are proposed, including JSONPath, XPath, and SPARQL. However, in our Search module, we prioritized the JSONPath implementation due to its flexibility and intuitive nature for querying JSON data structures which is the default encoding format for TDs.

The Things module provides comprehensive endpoints for TD operations, including creation, retrieval, modification, and deletion. Authentication mechanisms secure these operations, limiting modifications to authorized clients. Advanced querying options are available, allowing clients to filter and enrich TD listings with additional metadata and related information. The module's design adheres to RESTful API principles, offering intuitive endpoints and standard HTTP methods for simplified client integration and consistent interactions.

B. Authentication

The authentication module supports token-based authentication via username and password. This self-contained support model doesn't necessitate the use of an external service. However, we are actively working on refactoring this feature to adopt a more extensible approach. In the future, users will have the flexibility to select their preferred authentication mechanisms to suit their specific needs. For instance, they can opt for username and password authentication for smaller setups or utilize OIDC (OpenID Connect) for cloud-based deployments, ensuring enhanced security and user convenience.

C. Persistence

The Persistence module serves as an abstraction layer for data storage and retrieval. It utilizes the Knex.js[2] query builder to establish a robust connection with the PostgreSQL database and generate the needed queries. This module not only ensures the efficient management of user data and TDs but also adeptly handles TD lifecycle events. The *AbstractRepository* offers a generic blueprint for basic CRUDL operations promoting modularity and reusability, with specialized repositories like the *UserRepository* and *ThingDescriptionRepository* extending these operations for their specific needs. The *ThingDescriptionRepository*'s capability to process JSONPath queries is particularly noteworthy. To achieve this, a dedicated library[3] was developed to translate JSONPath queries into SQL/JSON Path, the language natively supported by PostgreSQL. This translation covers 90% of the language, and it's sufficient to support the querying of almost every TD. The *TDLifecycleEventRepository* is currently in-memory, but there are considerations to migrate to more persistent storage solutions like Redis to enhance scalability.

IV. REAL USE CASES

Despite being a newly developed software, Zion has already found extensive utilization as an important component in various real-world use cases spanning different IoT domains. Subsection IV-A elaborates on its role in supporting Structural Health Monitoring (SHM) systems, while Subsection IV-B introduces Zion's integration into a blockchain system with the objective of establishing a global IoT market.

A. IoT support for Structural Health Monitoring

Modern SHM deployments frequently entail installing diverse sensor devices capable of conducting long-term measurements. Consequently, there is a growing demand for dedicated software platforms that address scalability and interoperability requirements, ensuring the seamless integration of diverse sensors and enabling real-time monitoring and early detection of structural anomalies or potential issues in critical infrastructure assets. In such scenarios, IoT interoperability plays a core role by enabling seamless communication and interfaces of disjoint system actors. In the landscape of IoT platforms for SHM, MODRON [22], [23] rises as a versatile framework comprising software components designed to support applications spanning from sensor to cloud data acquisition and data management. This framework leverages the W3C WoT standard to integrate heterogeneous sensors and applications through a dedicated interoperability layer. Zion serves as a directory to register and manage TDs for sensors responsible for capturing accelerometer and acoustic emission data. In MODRON, every time a new WT is deployed, a component checks whether it is already registered in Zion, and if not, it proceeds to register it. This automated process allows for the discovery of WTs without human intervention. Additionally,

Fig. 1: MAC4PRO monitored concrete structure.

Zion is consumed by other MODRON components, which list and visualize the indexed WTs. Through the graphical user interface, users control which set of TDs and features they wish to visualize. This user input is translated into Zion queries. Figure 1 depicts one scenario of the MAC4PRO project in which MODRON enables the monitoring of the concrete structure. The image showcases a test of a shaking table applying seismic inputs to a concrete structure.

B. Blockchain Integration

The DESMO architecture presented in the recent study [24] enables a decentralized IoT global market where clients can retrieve IoT data from multiple data sources in a trusted manner through blockchain integration. This architecture finds its application in various domains, including urban scenarios related to noise pollution. By leveraging the DESMO system, entities can monitor and manage noise levels in real-time, ensuring adherence to regulations and enhancing the quality of life in urban areas. Within this framework, the data sources are indexed using specialized directory nodes that must comply with the W3C WoT Discovery specification. On the other hand, devices are described by a W3C WoT TD and must be registered on (at least) one of these TDDs, ensuring they can be located and utilized by the system. Zion is the official implementation, playing a critical role in enhancing the level of interoperability and performance. Its ability to perform complex JSONPath queries in a very efficient way is the key that allows the network to scale and become a real "global market" for IoT data.

V. PERFORMANCE ANALYSIS

We conducted a performance analysis to assess the scalability of the various implementations of TDDs compatible with
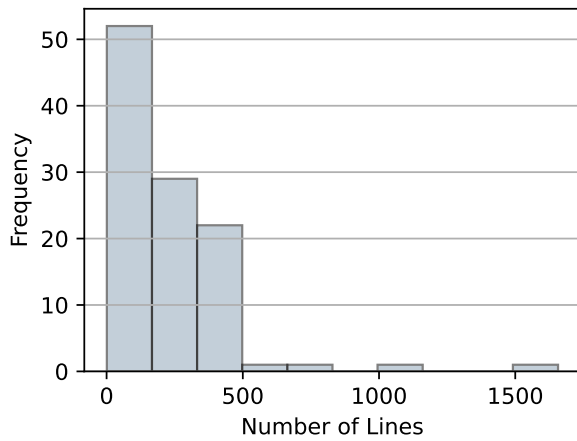
Fig. 2: Histogram that shows the distributions of WT TDs per number of lines

Fig. 3: Processing times for Zion and TinyIoT with y-axis in logarithm scale.

the W3C WoT Discovery standard; namely: Zion, TinyIoT, and WoT Hive. Our focus was on characterizing the scalability of the query time under different workloads, as it represents the most critical metric in this context. Unlike the less frequent operations of inserting, removing, and updating TDs, querying TDs occurs more frequently under real-world conditions – as searching and consuming internet content is a more common activity than the tasks of creating or removing content.

In our experimental setup, we performed experiments where we systematically varied the quantity of TDs stored in the database across different scenarios. In detail, we conducted experiments for 10, 100, 1,000, 10,000, and 100,000 TDs. We categorized the TDs into different complexity levels according to their number of lines: simple, medium, and complex. To guarantee a greater similarity with real-world scenarios, we distributed the categories of TDs stored in each experiment replication mimicking a Pareto distribution (the Pareto distribution is commonly employed to model the sizes of files on the internet) as Figure 2 depicts. Hence: 80% of simple TDs, 15% of medium TDs, and only 5% of complex TDs. We emphasize that the TDs utilized in the experiments represented real devices. They were collected from public repositories of official W3C WoT Events (e.g., PlugFest) and are publicly available in a GitHub repository[4]. Before each experiment, we populated the TDD with the TDs according to the specified distribution and complexity levels.

In each experiment, we conducted a hundred sequential calls to the TDD JSONPath search endpoint. For each call, we randomly select a query from the following options:

- `$[?(@.properties.lightColor)]`: this query searched for TDs containing the `lightColor` property.;
- `$[?(@.properties.lightColor && @.properties.brightness)]`: This query
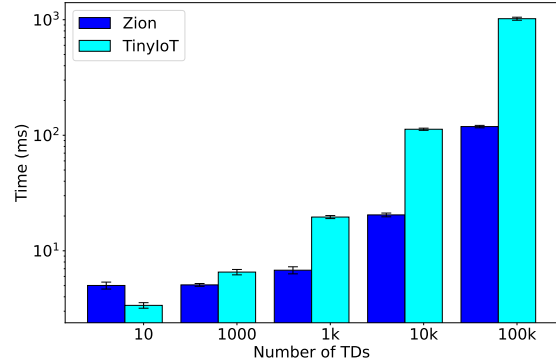
searches TDs featuring both the `lightColor` and `brightness` properties.
- `$[?(@.properties.sensorInformation)]. properties..state`: This query searches TDs with `state` as a sub-property within the `sensorInformation` property.property.

Preliminary experiments unveiled that all three queries have similar processing times. In each replication, we reset the database and re-populated it. Additionally, we utilized Mersenne Twister [25] algorithm as the pseudo-number generator, and we seeded it manually with a prime number for each replication. The manual seeding operation serves a dual purpose:

1) It ensures fairness in terms of the sequence of pseudo-random numbers generated when comparing different implementations. For instance, replication #12 of a specific experiment with Zion was seeded in the same manner as replication #12 of a corresponding TinyIoT replication.
2) It allows for the replication of the performance analysis by fellow researchers and developers, promoting transparency and reproducibility in our experimentation process.

The workload generator and the analyzed TDD were deployed in the same machine (12GB of RAM and an Intel Core i5-7200U CPU running at 2.50GHz with 4 cores). We containerized Zion and TinyIoT using Docker. Each experiment was replicated 30 times and asymptotic confidence intervals were computed at the level of 99%.

The results are depicted in Figure 3, note that the y-axis is in logarithm scale. We omitted WoT Hive from the graphs due to its unfeasible high processing time in all tested workloads. In the lowest workload scenario with 10 TDs, the average processing time was 0.94 seconds. This average increased to 1.85 seconds with 100 TDs and further extended to 8.61 seconds with 1,000 TDs. During all WoT Hive experiments, we encountered numerous error replies and experienced denial of service from the server. Our experiments support that WoT

---

[4]https://github.com/vaimee/tdd-workload-generator/tree/main/src/populate-db/examples-td

Hive is not suitable for the use cases defined in this study. Regarding the performance comparison for TinyIoT with Zion, we can note that the effect of increasing workload on Zion results in a steady, linear rise in its processing time. In contrast, TinyIoT experiences an exponential surge in processing time as the workload intensifies.

VI. Conclusion

In our work, we introduce Zion, an open-source Web Things directory implementing the W3C WoT Discovery Standard. It efficiently indexes devices and offers a discovery mechanism that searches device metadata. Zion provides a well-documented REST interface, allowing authenticated users to create, modify, update, and delete TDs. Regular users can list and search TDs using JSONPath queries. In our experiments, Zion outperformed other TDD implementations by scaling its service time linearly, while other TDDs either scaled exponentially or experienced crashes with increasing workloads. As future work, we plan to include a geospatial API and a tagging system. The geospatial API will enable device management and queries based on geographical locations, while the tagging system aims to provide mechanisms for categorizing and organizing TDs.

Acknowledgment

References

[1] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, Jun 2019.

[2] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "A comprehensive survey on interoperability for iiot: Taxonomy, standards, and future directions," *ACM Comput. Surv.*, vol. 55, no. 1, nov 2021.

[3] L. Sciullo, L. Gigli, F. Montori, A. Trotta, and M. D. Felice, "A survey on the web of things," *IEEE Access*, vol. 10, pp. 47 570–47 596, 2022.

[4] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of things (wot) architecture," W3C Recommendation, Apr. 2020, https://www.w3.org/TR/wot-architecture/.

[5] F. Montori, I. Zyrianoff, L. Gigli, A. Calvio, R. Venanzi, S. Sindaco, L. Sciullo, F. Zonzini, M. Zauli, N. Testoni, A. Bertacchini, E. Londero, E. Alessi, M. D. Felice, L. Bononi, P. Bellavista, L. De Marchi, A. Marzani, P. Azzoni, and T. S. Cinotti, "An iot toolchain architecture for planning, running and managing a complete condition monitoring scenario," *IEEE Access*, vol. 11, pp. 6837–6856, 2023.

[6] I. Zyrianoff, L. Gigli, F. Montori, L. Sciullo, C. Kamienski, and M. Di Felice, "Cache-it: A distributed architecture for proactive edge caching in heterogeneous iot scenarios," *Available at SSRN 4481856*.

[7] I. Zyrianoff, A. Trotta, L. Sciullo, F. Montori, and M. Di Felice, "Iot edge caching: Taxonomy, use cases and perspectives," *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 12–18, 2022.

[8] C. Puliafito, L. Gigli, I. Zyrianoff, F. Montori, A. Virdis, S. Di Pascoli, E. Mingozzi, and M. Di Felice, "Joint power control and structural health monitoring in industry 4.0 scenarios using eclipse arrowhead and web of things," in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2022, pp. 1–6.

[9] L. Sciullo, F. Montori, I. Zyrianoff, L. Gigli, D. Tinti, and M. Di Felice, "Designing a hybrid push-pull architecture for mobile crowdsensing using the web of things," in *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023, pp. 332–337.

[10] L. Hao and H. Schulzrinne, "Goldie: Harmonization and orchestration towards a global directory for iot," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[11] A. Cimmino, M. McCool, F. Tavakolizadeh, and K. Toumura, "Web of things (wot) discovery," World Wide Web Consortium (W3C), Proposed Recommendation, July 11 2023. [Online]. Available: https://www.w3.org/TR/2023/PR-wot-discovery-20230711/

[12] R. G.-C. Andrea Cimmino, "Wothive: Enabling syntactic and semantic discovery in the web of things," *Open Journal of Internet of Things (OJIOT)*, vol. 8, no. 1, pp. 54–65, 2022.

[13] F. Tavakolizadeh and S. Devasya, "Thing directory: Simple and lightweight registry of iot device metadata," *Journal of Open Source Software*, vol. 6, no. 60, p. 3075, 2021.

[14] I. Zyrianoff, L. Gigli, F. Montori, C. Kamienski, and M. D. Felice, "Two-way integration of service-oriented systems-of-systems with the web of things," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–6.

[15] I. Zyrianoff, A. Heideker, L. Sciullo, C. Kamienski, and M. Di Felice, "Interoperability in open iot platforms: Wot-fiware comparison and integration," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2021, pp. 169–174.

[16] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, and M. Di Felice, "From cloud to edge: Seamless software migration at the era of the web of things," *IEEE Access*, vol. 8, pp. 228 118–228 135, 2020.

[17] Y. Zhou, S. De, W. Wang, and K. Moessner, "Search techniques for the web of things: A taxonomy and survey," *Sensors*, vol. 16, no. 5, 2016. [Online]. Available: https://www.mdpi.com/1424-8220/16/5/600

[18] Z. Ding, Z. Chen, and Q. Yang, "Iot-svksearch: a real-time multimodal search engine mechanism for the internet of things," *International Journal of Communication Systems*, vol. 27, no. 6, pp. 871–897, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.2647

[19] L. Hao, V. Naik, and H. Schulzrinne, "Dbac: Directory-based access control for geographically distributed iot systems," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 360–369.

[20] M. R. Faheem, T. Anees, M. Hussain, A. Ditta, H. Alquhayz, and M. A. Khan, "Indexing in wot to locate indoor things," *IEEE Access*, vol. 11, pp. 53 497–53 517, 2023.

[21] D. Ottolini, I. Zyrianoff, and C. Kamienski, "Interoperability and scalability trade-offs in open iot platforms," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 1–6.

[22] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, F. Zonzini, L. De Marchi, M. Di Felice, A. Marzani, and T. S. Cinotti, "Modron: A scalable and interoperable web of things platform for structural health monitoring," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021, pp. 1–7.

[23] L. Gigli, L. Sciullo, F. Montori, A. Marzani, and M. Di Felice, "Blockchain and web of things for structural health monitoring applications: A proof of concept," in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 699–702.

[24] L. Gigli, I. Zyrianoff, F. Montori, C. Aguzzi, L. Roffia, and M. Di Felice, "A decentralized oracle architecture for a blockchain-based iot global market," *IEEE Communications Magazine*, vol. 61, no. 8, pp. 86–92, 2023.

[25] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, p. 3–30, jan 1998.