

Article

A Generic Internet of Things (IoT) Middleware for Smart City Applications

Zulfiqar Ali ¹, Azhar Mahmood ^{2,*}, Shaheen Khatoun ³, Wajdi Alhakami ⁴, Syed Sajid Ullah ^{5,*},
Jawaid Iqbal ⁶ and Saddam Hussain ⁷

¹ Department of Computer Science, Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad 44000, Pakistan

² Faculty of Computing, Capital University of Science and Technology, Islamabad 44000, Pakistan

³ School of AI and Advanced Computing, Xian Jiaotong Liverpool University, Suzhou 215000, China

⁴ Department of Information Technology, College of Computers and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia

⁵ Department of Information and Communication Technology, University of Agder (UiA), N-4898 Grimstad, Norway

⁶ Department of Computer Science, Capital University of Science and Technology, Islamabad 44000, Pakistan

⁷ School of Digital Science, Universiti Brunei Darussalam, Jalan Tungku Link, Gadong BE1410, Brunei

* Correspondence: azhar.mahmood@cust.edu.pk (A.M.); syed.s.ullah@uia.no (S.S.U.)

Abstract: The Internet of Things (IoT) is one of the key components of the ICT infrastructure of smart cities due to its great potential for intelligent management of infrastructures and facilities and the enhanced delivery of services in support of sustainable cities. Smart cities typically rely on IoT, where a wide variety of devices communicate with each other and collaborate across heterogeneous and distributed computing environments to provide information and services to urban entities and urbanites. However, leveraging the IoT within software applications raises tremendous challenges, such as data acquisition, device heterogeneity, service management, security and privacy, interoperability, scalability, flexibility, data processing, and visualization. Middleware for IoT has been recognized as the system that can provide the necessary infrastructure of services and has become increasingly important for IoT over the last few years. This study aims to review and synthesize the relevant literature to identify and discuss the core challenges of existing IoT middleware. Furthermore, it augments the information landscape of IoT middleware with big data applications to achieve the required level of services supporting sustainable cities. In doing so, it proposes a novel IoT middleware for smart city applications, namely Generic Middleware for Smart City Applications (GMSCA), which brings together many studies to further capture and invigorate the application demand for sustainable solutions which IoT and big data can offer. The proposed middleware is implemented, and its feasibility is assessed by developing three applications addressing various scenarios. Finally, the GMSCA is tested by conducting load balance and performance tests. The results prove the excellent functioning and usability of the GMSCA.

Keywords: smart city; Internet of Things; middleware; service-oriented; microservices



Citation: Ali, Z.; Mahmood, A.; Khatoun, S.; Alhakami, W.; Ullah, S.S.; Iqbal, J.; Hussain, S. A Generic Internet of Things (IoT) Middleware for Smart City Applications. *Sustainability* **2023**, *15*, 743. <https://doi.org/10.3390/su15010743>

Academic Editors: Kamal Bechkoum and Martin Wynn

Received: 31 October 2022

Revised: 22 December 2022

Accepted: 28 December 2022

Published: 31 December 2022



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate between networks and devices. The Internet was mainly a network of computers. The subsequent development is the Internet of Things (IoT), where things communicate in a network. It will change our lives the same way the Internet did. However, it involves the communication of things with other things. One example of such communication is when an alarm is switched on by an object sensor or a motion detector: this communication will occur without humans' intervention. IoT is a diverse and vast development area. It can be defined as follows: "IoT is the internet

of things connected with each other using smart sensors and can operate without the intervention of humans" [1,2].

Numerous studies have been carried out in the area of IoT, and many challenges and issues have been highlighted [1,3–7]. Some of these issues are the standardization of protocols for IoT devices to communicate, the lack of description language for services to make them compatible with different implementation environments, the interoperability of IoT devices, the storage and processing of data streams generated by IoT devices, the lack of online machine-learning techniques to process the data in real time, data curation for faster processing, the design of service-oriented architecture, the implementation of context-aware computing, the security and privacy of data generated by IoT devices, the integration of IoT in current information & technologies environments, and the scalability and extensibility [1] of IoT applications to incorporate more devices.

Due to the fact of that there are many challenges that developers face during the development of new applications based on IoT, a middleware solution can be helpful in this regard. Authors in [8] realize the importance of middleware for developing applications in various domains, including applications based on IoT. They presented the advantages such as speedy development of new software and uncovering issues at the early stage of development.

The diversity of IoT and its large number of applications and devices poses the problem of integration of these applications and devices. The devices and applications need to speak the same language. As a result, standard middleware needs to be upgraded in order to integrate multiple devices in one place under a unified language. The critical challenge is the implementation and standardization of middleware for IoT systems [1]. Different surveys have been carried out to evaluate the state-of-the-art middleware and compare different works [3–6,9]. A review of concepts, frameworks, and key technologies is presented in [10]. The authors have identified the different layers in the architecture of frameworks for smart city applications and discussed the responsibility of these layers. Finally, they have tabulated a summary of smart city domains, services, application features, IoT, and sensing technologies involved in these domains and real use cases with operational applications in different parts of the world. Many solutions have been proposed to address the challenges in IoT-based applications, but there is no standard solution available yet. The proposed middleware focused on one or the other area of applications, and not a single middleware solution is proposed that satisfies all application requirements of even a single application domain. Benchmarks are defined, and generic application requirements are identified [9,11], so a solution is required to fulfil all the requirements to entirely address the generic needs along with the few specific application domain requirements, which are somehow different for each application domain.

One of the potential application domains for IoT is the applications for a smart city. Many researchers have proposed frameworks, and several surveys have been conducted to evaluate the performance of these frameworks [3–6]. For instance, in [9], the authors have evaluated and surveyed some middleware for different domains based on IoT. The authors have categorized the middleware into different categories based on their underlying architecture and tabulated the challenges they address. The authors have also listed the benefits and challenges addressed by middleware carried by the underlying architectures of the middleware. There are several other works conducted in the field of IoT, such as graph-based M2M optimization in IoT environment [12], cooperative cognitive intelligence for the Internet of vehicles [13] and defining human behaviors using big data analytics in social Internet of things [14]. An effort toward defining a generic architecture for IoT is made in [15].

There are a few other paradigms proposed for IoT which could be used as a component of IoT middleware. For instance, [16] proposed CA4IoT, focusing on context awareness for IoT systems. The work is intended to present a framework for maintaining the sensor information with context awareness so that sensor data can be provided to the users on demand. On the other hand, FIWARE-Lab is proposed in [17]: it is intended for resource

and service management in the cloud federation to support the developers in building future internet applications. By using the FIWARE-Lab project, different service providers come into a contract where they share their services for the development of new applications by following some terms. Likewise, Smart M2M gateway-based architecture is proposed for managing billions of devices and endpoints [18]. Sensor markup language (SenML) is used for the measurement of sensor readings and actuator commands. RESTful web services are used for the gateway. APIs and this gateway are then used to manage the devices and endpoints.

Although many existing studies have tried to propose additional components that could be plugged into any middleware to address one or the other key challenge, they still have many limitations. For instance, it is unclear how the repositories will be maintained and whether the information should be maintained in a database or flat files. Furthermore, none of them addresses all the core challenges and generic big data application requirements for any domain. Hence, there is a need for a comprehensive solution that addresses all the core identified challenges with a system which should be flexible enough to incorporate the services that would emerge at later stages, along with the flexibility of changing the services over time upon need. Therefore, in this work, a complete framework for the end-to-end architecture of IoT-based applications and the proposal for middleware for IoT-based applications, GMSCA, is presented. The challenges of data acquisition, scalability, heterogeneity, flexibility, extensibility, data analytics, and security are handled through different components, as discussed in Section 3.3.

This paper aims to review and synthesize the relevant literature to identify and discuss the key challenges pertaining to IoT middleware. It explores the opportunity to augment the information landscape of IoT middleware with big data applications to achieve the required level of services in support of sustainable cities. In doing so, it proposes a novel IoT middleware for smart city applications, GMSCA, which brings together many studies to further capture and invigorate the application demand for sustainable solutions which IoT and big data can offer. The proposed framework provides a platform to develop IoT-based big data applications with data processing and analysis capabilities. As a proof of concept, the proposed middleware is tested with the development of three applications in the context of smart cities. In the basic implementation, the middleware ingests the dataset from the users, and then its functions in the middleware prepare it for appropriate machine-learning models. Data is then run through different machine-learning algorithms, and the best model is chosen and deployed as a service for predictions. Hence, the proposed framework, which can be replicated, tested, and evaluated, will add depth to field studies and provide a foundation for researchers to draw on for analytical insights in future research.

The rest of the paper is organized as follows: Section 2 discusses the relevant conceptual and theoretical constructs that make up the study. An overview of the methodology to elaborate the research process, followed by a relevant literature review, critical evaluation, and the proposed middleware, GMSCA, are presented in Section 3. The implementation detail of GMSCA is presented in Section 4. Details of applications developed using the proposed middleware are discussed in Section 5. Load balance and performance testing are discussed in Section 6. Finally, we conclude and provide insight into future work in Section 7.

2. Background

Urbanization has been increasing at a swift pace throughout the globe [19]. This urbanization has been creating lots of issues such as traffic congestion, environmental changes, lack of green areas, and lack of health and educational facilities. It has also made management a difficult task for the city management authorities. To address such growing issues, smart solutions are required, which can help to manage all the resources efficiently and provide higher living standards to city inhabitants. A smart city [20] seems to be a suitable solution for this problem and can help to solve the issues using technology and infrastructure.

Different technologies are used to make a smart city application. Each of these technologies has a specific role in making the smart city a reality. Sensors are the core technology used for creating smart city applications. These are deployed in the environment to monitor and send the desired environmental data to the server/cloud using the gateways to make the application logic aware of the environment. With the fast evolution of sensor technology, a huge array of sensors is available on the market. Temperature sensors, humidity sensors, camera sensors, and motion sensors are just a few basic examples of the available sensors. Since these sensors are of different natures and send different types of data, there are different interfaces and different access methods. This fact introduces the challenge of heterogeneity. These sensors are then required to be accessed by different applications and operating systems, thus introducing the interoperability issue.

Smart City is an extensive domain comprised of many subdomains. Smart health monitoring, intelligent transportation, intelligent traffic lighting, smart water, smart grid, and smart education are just a few examples of sub-domains. Thus, Smart City has a very vast scope and, as a result, may not be developed in one go but rather step by step. This fact highlights the scalability, flexibility, and extensibility challenges in the domain. Indeed, services may require to be added or to be changed over time due to new emerging requirements. The scalability issue arises when the available resources are consumed, and the requirements for more resources are still intact.

There are some challenges that must be handled to make reliable and functioning smart city applications. Table 1 lists a few of the important challenges. Besides the listed challenges, there might be many other challenges such as context awareness, quality of service, data processing, and data analytics. There may be different strategies to address the challenges. It might be handled using the algorithms and developing the components in a way which is not cohesive and tightly coupled so that we can make changes in the components, when and where required. Another method that is mostly used and more reliable is architecture, such as services-oriented architecture or microservices-based architecture, which supports and helps to cope with the listed challenges.

Table 1. Challenges for the smart city applications.

Challenges	Reason
Data Acquisition	Data from sensors/hardware is required. Data acquisition is therefore a must and represents the core of smart city applications.
Heterogeneity	Different devices, sensors, and applications with different data formats introduce heterogeneity.
Flexibility	Previous components and services may be required to be altered over time due to new/changing requirements.
Extensibility	More components and services may be required to be added, so applications must be extensible.
Scalability	The large scope and ever-increasing requirements to address more service requests may exhaust the resources. As a result, applications must be scalable.
Security	Users' data must be secure and sometimes private.

Due to these challenges and issues, middleware is required to avoid every smart city application having to deal with these issues right from the beginning. Instead, the applications would be developed on top of the middleware, which solves the issue of creating everything from scratch and provides application developers with a central repository of services with a standard communication method. Fog is a technology that can be used to increase the performance of smart city applications [21]. It resides at the edge of the network and works like the Cloud. A subset of the services is placed in Fog to serve the network with better performance, and, in case of the absence of the required service, the request is forwarded to the Cloud to provide the requested service. Fog is also used for

processing data and sending over only the required data to the Cloud. It not only offloads the work on the Cloud server but also increases the speed of response to the user. It helps to decrease the latency and also makes applications location-aware. An example scenario is getting the room temperature using the temperature sensors employed in the room. In this case, a Raspberry Pi server, which also works as the gateway for IoT devices, can process the readings from different sensors and compile an aggregate at the gateway level. Another advantage of Fog is that it increases the system's scalability by providing the services from Fog and sending less traffic to the Cloud, thus saving bandwidth. In [21], the authors have proposed a framework to deal with the faults that may occur when fog architecture is used.

Smart city applications are developed using different architectures, which all have their advantages and disadvantages. Some of the architectures include distributed architecture, component-based architecture, service-oriented architecture, and microservices architecture. Service-oriented architecture is the most famous architecture, and many researchers have used it. It brings many advantages such as scalability, flexibility, and extensibility, thus making smart city applications easy to handle with fewer challenges. In this architecture, different services are created and deployed on the server, and all requests are handled by a service broker to register, discover, or invoke the required service.

Micro-service-based architecture is one of the new architectures which have evolved from the best industry practices. In this architecture, components are developed as small services that can be self-contained, managed, and deployed independently. All services can be combined using some interface to make new services, and each could be scaled or changed individually without affecting the others. In this way, this architecture helps with scalability and flexibility. More services could be added as required, addressing the challenge of extensibility.

In this paper, we aim to provide a topical literature review of the existing IoT middleware architectures to identify strengths, weaknesses, and challenges of such middleware for the development of smart city applications. The main focus is to derive from the literature analysis and further capture the information landscape of IoT architectures to develop a generic middleware addressing most of the key challenges in developing and deploying a specific smart city application.

3. Methodology

The purpose of this study suits a mix of the following research approaches: (1) topical literature review, (2) critical analysis to achieve different yet related objectives, and (3) proposal of a generic middleware architecture addressing key challenges. The topical literature provides a survey of related work covering the state-of-the-art IoT middleware architectures, related applications, issues, knowledge gaps, and research opportunities. In the critical analysis, a gap analysis is performed concerning the lack of research on developing integrated frameworks for smart, sustainable cities as a holistic urban development approach, which provides a basis for the proposed generic framework. Finally, a novel IoT middleware for smart city applications, GMSCA, is proposed by bringing together many studies to further capture and invigorate the application demand for sustainable solutions which IoT and big data can offer. This is to be derived additionally from the literature analysis focused on smart city applications and the underlying IoT middleware, data processing platforms, and computing models based on the IoT and big data analytics. Each research approach is further discussed in the following sections:

3.1. Literature Review

This section provides a detailed review and synthesis of the technical literature on IoT middleware architectures. Different IoT-based middleware has been proposed in the past. These are based on different architectures, addressing different challenges, and targeting different application areas. In [7], the authors have highlighted the challenge of interoperability/heterogeneity among the connected devices and proposed a framework using open-access technologies and frameworks to address this challenge. In [22], the

authors have categorized the middleware into four categories, which are publicly traded IoT Cloud platforms, open source IoT Cloud platforms, developer-friendly IoT Cloud platforms, and end-to-end connectivity IoT Cloud platforms. The authors have compared the basic features, sensing features, communication features, and application development features of the features lying in each of the aforementioned categories and finally defined criteria for the selection of middleware for specific application needs. In [9], authors have surveyed different middleware and listed the advantages and challenges of specific architectures. Every architecture has a different set of challenges and architecture-specific advantages. The reviewed middleware are categorized based on their respective architectures in the following sections.

3.1.1. Component-Based Middleware

In component-based architecture, a solution is built by several small components that work together to make a complete functional solution. Many such middleware and platform proposals are based on component-based architecture, such as:

SGeoL, a platform for integrating heterogeneous data and developing smart city applications based on FIWARE middleware, is proposed in [23]. The platform is based on the component-based architecture, and different components, including security components, broker components, data integration, and analytic components are developed. These, in turn, address the challenges of security, extensibility, flexibility, and heterogeneity for the applications based on IoT devices.

In [24], component-based middleware ‘Device Nimbus’ is proposed, which addresses the challenges of context awareness, scalability, flexibility, extensibility, lightweight, standard compliance, and resilience in the field of IoT and middleware requirements. A minimum viable application is built on top of the middleware to show its effectiveness and feasibility. Environment sensors and social networks, e.g., Twitter and Facebook, are used to answer a few questions to identify people’s fitness and health patterns. MinT is another example of component-based middleware [25]. The authors addressed the problem of energy-efficient processing due to high energy constraint devices in IoT. Validation and performance evaluation of MinT were conducted using experiments. The average throughput is used as the evaluation criteria, and the middleware is compared with two other middleware, namely nCoap and Californium. MinT outperforms the aforementioned by achieving the maximum throughput of 8900 requests per second.

Due to the diverse nature of devices being used in the field of IoT, there are many different data formats, sources, and access methods. To address this complex situation, the DAQ middleware is proposed [26]. It deals with the heterogeneity of the different data sources and data formats and makes data acquisition simple and efficient. There are two algorithms in DAQ that make the acquisition better: the parallel acquisition algorithm and the heuristic algorithm. The parallel acquisition algorithm is most effective when an acquisition is carried out from multiple interfaces in parallel. However, it is less efficient when the data acquisition is made under the same interface. The results of the acquisition by serial, random acquisition, and heuristic-based acquisition were compared, and the heuristic-based acquisition appeared to be the best among competitors. GAMBAS was created to address the challenges of data acquisition, data distribution, and data integration [27]. GAMBAS has a component-based architecture and was proposed for generic applications in the domain of smart cities. The usability and implementation are showcased using two applications. These are Voiceprint Launcher and Linked Weather. Voiceprint Launcher is an application run locally on phones, although substantial parts and features of the middleware are implemented. It launches the applications on a phone by voice command. Linked Weather, on the other hand, shows how the middleware can help to manage the data and how remote communication works. These two applications help understand and validate the data acquisition, security and privacy, data processing, and communication components. A large-scale demonstrator is being developed in the domain of transport

in the city of Madrid. It will provide real-time transit information. This application will further strengthen the effectiveness of GAMBAS in a smart city environment.

Another end-to-end Security Middleware was proposed for flexible and configurable security, depending on the application's needs [28]. Static Pre-Shared Keys (PSKs) are realized as an efficient choice for faster processing and lightweight security implementation. Moreover, the session resumption technique is used for faster data exchange and retrieval of the trust base. The current middleware version does not support short-lived sessions through the session resumption technique. In its current implementation and due to the use of the session resumption, the trust state is required to be maintained in the network amongst the IoT devices, and a short-lived reputation is not supported. A short-lived reputation or trust state must be supported to avoid maintenance of a trust state for longer periods and to deal with the transient devices. A minimum viable implementation of the middleware has been used on a GENI Cloud testbed. The results are shown for the memory footprint and time for the security association. The results have proved that the memory requirement for the chosen scheme is less in comparison to the other schemes, while the security association time is about 80 ms and the session resumption takes around 40 ms.

In [29] Mint-I, an improved version of the MinT [25] middleware is proposed to address the problem of thread pools being created as per CPU cores and the resulting waste of resources when the created threads are more than the throughput required. A Variable Thread Adjustment (VTA) is proposed to save resources and efficiently utilize resources with improved performance. VTA looks for the throughput of the requests received from IoT devices and the queue size of the requests received to calculate the number of threads required. The proposed middleware's limitation lies in that when it starts aggregating peripheral information from devices when the client requests information from it, the increase of requests' energy consumption increases significantly due to the aggregation cycle by the middleware. The aggregation of information from sensing devices that do not update information frequently causes transmission and excess energy consumption delays, which can be avoided by efficient management methods to control these aggregation cycles.

An architecture to provide support for the resilience level of IoT-based applications for a smart city is presented in [30]. The proposed architecture works towards the improvement of resilience in smart city applications by using technologies such as cloudlets (Fog) and core Cloud. It improves resilience along with scalability. Discussion is provided using hypothetical examples and two scenarios, and explaining how the proposed system will work towards the recovery of faults occurring during the application. One scenario is that of a failure detected by the monitor, then the protection and recovery module works with other modules, such as the placement and migration module, to recover the faulty service or other actions to overcome the fault. Another example is considered when the drop in quality is generated, and the monitor sends the notification that a drop in QoS is generated by the QoS module. Then, the protection and recovery module takes the necessary actions, such as finding an alternative path with lower latency to overcome the issue. The testing of the proposed architecture is not performed in a real-life smart city, whilst it should be in order to affirm the evaluation of the resilience of the system. A software architecture that is loosely coupled and message-oriented in [31]. It is driven by semantic OWL messages. For implementation purposes, a use case is provided with the theoretical explanation of sensors and software components with data flow and the processing required to detect a bedsores disease risk in older people. One of the limitations of the architecture is that it was not evaluated in the clinical environment, although it should be for reliability and usability testing purposes. Moreover, the proposed middleware architecture must be implemented to affirm its suitability.

3.1.2. Distributed Middleware

In this architecture, the solution is built and deployed in a distributed manner. Both hardware and software may span multiple locations and may work independently for some functions or be coordinated to perform a specific task.

A smart and semantic middleware (SMArc) is proposed in [32]. The architecture is distributed, and it targets energy management in the smart city domain. Interoperability, scalability, and heterogeneity are the challenges addressed by the authors. The middleware is evaluated for performance in service registration and service completion. The authors have defined two types of service requirements: simple services and composed services. Results for service registration range between 274 ms and 651 ms, for service completion between 257 and 1227 ms for simple services, and between 259 and 1230 ms for composed services. In [33], Civitas is proposed, which is a distributed and object-oriented middleware. The middleware does not target a specific application and is proposed as a generic middleware to target a variety of applications in the smart city domain. One of the benefits of Civitas is that it has a commonsense reasoning engine, and though limited, it definitely gives a framework to incorporate more functionality in this context. The challenges such as scalability, heterogeneity, and privacy issues are identified as the main challenges that must be addressed whilst developing applications for smart cities. To validate the proposed middleware and to show how to use it, the authors have conducted a case study in which they showed how to track the license number plates of a vehicle. How the current proposed framework could be used for service development and how it fits in the picture of IoT is also indicated, as well as how the intelligence part in the middleware could be useful in this scenario. One of the areas for future direction is the design of a model for the security and privacy of the data of citizens.

An independent Distributed Data Service (DDS) [34] is created that could be plugged into many IoT middleware. It is used for data collection, data processing, and data aggregation, thus relieving the middleware and helping to avoid the re-invention of the wheel, as it is a must-have part for each of the smart city applications and thus becomes a necessary requirement of the middleware for IoT. DDS is integrated with two middleware, UIoT and Kaa, to demonstrate its feasibility and effectiveness. The results of data collection and ingestion are listed. The results showed that DDS has the best message ingestion when asynchronous mode is used as opposed to when a synchronous mode is used with the UIoT. The results of data collection show that it outperforms the Kaa data collector, proving its effectiveness as compared to the other middleware's data collection capability. A distributed stream reasoning system using the processing of large volumes of data with scalability is presented in [35]. The authors have also evaluated the proposed system, and the results for performance are shown. The stream reasoning system is implemented using Apache Kafka and Apache Storm, and middleware SOUL. The results show the capability of the system to be about 10,000 messages per second. Moreover, the transmission time is the least when the stream data transmission is 10,000 per second. The system is tested and evaluated using simulations and not tested on the real-time streaming of big data.

In [36], the authors proposed a FogFlow middleware with a published and subscribed paradigm for service brokers. It is based on the programming model to gain openness and interoperability for the developers. It leverages edge and cloud technologies for data processing with lower bandwidth and decreased latency. Three use cases are described, along with the implementation of one of the three use cases. An application for the anomaly detection of power consumption in retail stores was developed. Different results demonstrate the achieved performance and throughput as well as lower latency values. It has higher throughputs for the service discovery queries and responses. For different propagations, latency is shown for different test cases. For the same broker, the average latency is about 0.7 ms, while for the different brokers in the same data centre, it stays under 50 ms, and for different brokers and different data centres, it reaches around 430 ms.

To improve resource utilization in the smart city, an agent-based middleware framework based on distributed CPS is proposed in [37]. As per the claim of the authors, this middleware helps improve the reliability of communication in a smart city environment due to the use of agent technology, which in turn uses the linear computation model and resolves the data source downtime issues.

A middleware CHARIOT [38] to address the issues of resilience in Edge computing-based IoT systems is proposed. It implements a three-layered distributed architecture. The layers are: system description language, data storage layer, and management engine layer. CHARIOT makes use of Satisfiability Modulo Theories (SMT) solvers to compute optimal system (re)configurations dynamically at runtime. Evaluation is done by runtime implementation of CHARIOT using the Smart Parking System use-case scenario. CHARIOT's runtime implementation architecture consists of compute nodes comprising the layered stack. Each CHARIOT-enabled compute node hosts two platform services: a Node Monitor and a Deployment Manager. The Node Manager assesses the liveness of its specific node, whereas the Deployment Manager manages the lifecycle of applications deployed on a node. CHARIOT's Node Manager is implemented as a Zookeeper client that registers itself with a Monitoring Server. The Deployment Manager is implemented as a ZeroMQ subscriber that receives management commands from a Management Engine. A Database Server is an instance of a MongoDB server. MASSIF platform [39] is a distributed platform developed for intelligent data processing. It focuses on reactive and real-time data processing that complies with the objectives such as semantic annotation of data using the developed ontology, knowledge extraction, high-level workflows, real-time processing of data, extensibility, scalability, and performance. The proposed middleware has been validated using two use cases, which are organizing the home care cloud platform (OCCS) and real-time automation of media production for interactive radio and conferences (RAMP). Both use case projects use low-level data and background knowledge to extract new high-level knowledge. The middleware is limited in scope and does not support the processing of large streams of data, such as those from Twitter or Facebook. The authors plan to incorporate stream data processing techniques for less complex reasoning scenarios. Moreover, the authors plan to investigate machine-learning techniques for unknown sensor data. To make the platform more scalable, the authors decided to investigate load-balancing techniques and the automated duplication of services.

3.1.3. Service-Oriented Middleware

In this architecture, software platforms are built to provide different services to the applications. Functions are developed and deployed, and access to the functions is granted using Application Programming Interface (API) calls. Different services can coordinate with each other to perform a specific task. All the services built are accessed using the Enterprise Service Bus (ESB). It is the most used architecture in the field of IoT middleware. The middleware included in this category are discussed below:

MiSCi, an autonomic reflective middleware for smart cities, is presented in [40]. The Middleware is based on intelligent agents that can be adapted to the existing dynamism in a city. The architecture of the MiSCi is based on web services and multi-agent systems. These agents create ontologies to solve problems that arise in different situations. MAS makes use of the MAPE-K loop for monitoring, analyzing, planning, and executing the necessary actions in a particular situation.

A cloud-based intelligent car parking middleware ABC&S is presented in [41]. It is based on the always-connected and best-served paradigm. Its architecture is service-oriented, and the authors have created an application for the intelligent car parking system. The application contains three layers, which are the cloud tier, the web service tier, and the mobile application tier. The evaluation of the application and middleware is done using the performance of the response rate. The average response rate was less than one second, according to the results.

Rimware [42] is also a service-based middleware like a few other middleware [41,43–46]. The two primary challenges addressed by the authors are the need for devices to have a secure connection to the Cloud by alternative gateways in the absence of a primary smartphone or set-top boxes and for the Cloud to model the device capabilities. The authors have implemented the Rimware as Blue Rim to validate the cross-interoperability of the device for different applications. CotWare is another service-oriented middleware targeting the entire domain of

smart cities [43]. It deals with the requirement for integration of Fog and Cloud platforms to address the flexibility, extensibility, heterogeneity, and scalability in smart city applications. The authors have created a lab setup to test and validate the service implementation offered by the proposed middleware. The middleware is evaluated using the performance of the communication calls. The average time for local calls turned out as 500 ms, whereas remote calls took around 3000 ms. The service lookup ranged between 1300 ms and 2400 ms.

The service requirement of smart grids is increasingly changing and at a fast pace. The applications built for the current smart grids are heterogeneous in nature due to the diversified service request. The development of these applications is cumbersome because of the absence of flexible middleware supporting the development of heterogeneous applications. To address this issue, a Service Oriented Middleware is proposed [44]. It targets the smart grid application area of the smart city domain. The authors have evaluated the middleware using lab experiments and have shown that the proposed solution outperformed the previously proposed paradigms named TDM (Time Dependent Middleware) and PAM (Power Aware Middleware). The future plans of the authors are to apply the same middleware to the practical smart grid environment. In [45], SmartCityWare is proposed, which is a service-oriented middleware, generic in nature and targeting smart city domain areas. The authors have tried to achieve the proper integration and efficient utilization of CoT and Fog Computing using the service-oriented middleware (SOM) approach and tried to resolve some of the challenges of developing and operating smart city services. Due to the research work carried out and the development of SmartCityWare, several challenges in the smart city domain are addressed: these include extensibility, flexibility, security (by implementing the security service in the proposed middleware), QoS (by introducing fogs), better communication among the systems and devices (fog–fog, fog–cloud, device–fog calls), and heterogeneity (by using different types of devices and systems), among others.

A data processing middleware based on a service-oriented architecture for integration and fusion of multi-source heterogeneous data is proposed in [46]. The middleware is tested in a practical environment using environmental sensors. Thirty sensor nodes are deployed in three rooms to monitor the ambient temperature. These nodes are further connected to a base station and themselves connected to a cluster of four common PCs running Ubuntu. The authors claim that experiments have shown the effectiveness of the middleware, although no results are shown.

A microservice-based platform SAVI-IoT is provided in [47] that deals with autonomic management, programmability, distributed heterogeneity, security, and privacy. The autonomous management system is implemented using the MAPE-K loop. The authors have used the edge along with the Cloud to create the platform. The proposed platform SAVI-IoT is a programmable and self-managing platform based on microservices. The platform is generic enough for different IoT use cases. Big data compatibility, in-place data processing, high-level programmability, elasticity, fault tolerance, and auto-scalability are among the prime features of the presented IoT platform. In [48], the authors have adapted and ported a lightweight thread-based middleware LISA, built for RIOT, to an event-based Contiki OS. They accomplished it by defining and handling a set of events in order to communicate with user applications. To demonstrate the middleware and observe the experimental results for this work, the Cooja simulator is used. Cooja is a network simulator that allows users to test various network configurations and applications on different hardware platforms. The authors used an MSP430F5438 microcontroller from Texas Instruments. It has 256 KB of flash and 16 KB of RAM. It is an ultra-low power microcontroller with CC2420 or CC2430, 2.4 GHz 802.15.4 radio, which is compliant with the IEEE 802.15.4 standard. Two main limitations of the middleware are, on the one hand, the overuse of radio and CPU time from the routing of the user messages and the introduction of the memory overhead. On the other hand, the second limitation is the lack of support for more protocols. Currently, only 6LoWPan is supported in the first version of the adapted middleware.

In [49], the authors proposed an event-based service-oriented middleware (based on LinkSmart) and a heating, ventilation, and air conditioning (HVAC) control strategy

to monitor and manage the energy consumption in buildings and public spaces. It is found that applying the user-centric approach along with service-oriented and event-based middleware helped in achieving energy savings and also helped in interoperability, integration and heterogeneity. The middleware was implemented and evaluated for its effectiveness. The system is deployed with sensors in a historical building. Two rooms are considered test cases, whereas another two rooms with the same structure and dimensions are considered reference rooms. The results showed that the deployed system helped in energy savings for heating and cooling in the test rooms compared to the reference rooms while providing the same level of comfort. The use case described by the authors for the evaluation is not enough to affirm the stated claims. More rigorous testing is required to prove the suitability of the middleware as the devices and optimization goals could differ significantly when it is deployed on a larger scale.

A service-oriented autonomous middleware is proposed in [50] to address IoT-based application requirements by incorporating MAPE-K loop (Monitor, Analyze, Plan, Act, using stored Knowledge) and a multilayer context model. In [51], the authors developed a service-oriented middleware TinyCO to address the challenge of interoperability. The proposed middleware can identify the underlying heterogeneous network based on TinyOS and Contiki operating systems and converts the generic request to the destination-specific request. For implementation purposes, the authors have created two networks, one based on Contiki OS and another based on TinyOS. For the nodes, they used TelosB mote, which supports both selected operating systems. The middleware is used to identify which specific TelosB is connected to which part of the network, and then the application request is converted to a format understandable by the destination network. The solution is limited as it is given only for two operating systems-based networks, which are TinyOS and Contiki. More diverse networks must be incorporated to make the solution even more reliable and usable for a greater number of use cases.

A middleware-based infrastructure [25], which is scalable (using separate databases with proxies (APIs) for different data sources) and can handle heterogeneous data sources for energy consumption management and visualization and simulation at the district level, is provided. The infrastructure used the SEEMPubS middleware along with DIMCloud (District Information Model Cloud). Other technologies used in infrastructure are Building Information Models (BIMs), System Information Models (SIMs), Geographic Information Systems (GISs), and an ontology manager. An ontology manager is used to provide the semantic description of the models. A use case is described to explain the flow of data in the proposed infrastructure. The test results of the deployed system in the real district are not shown to affirm the claims and usability of the infrastructure.

3.1.4. Microservices-Based Architecture

It is one of the latest architectures and has several advantages. It could be used as a service-oriented architecture. Moreover, different microservices are developed and deployed independently. These can be scaled individually and can be modified or removed without affecting the other services. It does not require one ESB to provide the services. The reviewed proposals that were developed using this architecture are as such:

A microservices-based software infrastructure is developed for the management of energy in a smart city by using building models, energy profiles, and grid models [52]. It addresses the challenges of the integration of heterogeneous data sources. The infrastructure is implemented in the real district for experimentation, and a control policy for energy management is applied. Three rooms in a school building are used for test cases, and around four thousand sensors were installed in the entire district to monitor and manage energy usage. Based on the microservices architecture, this software infrastructure is scalable and can integrate heterogeneous devices. The results have shown that the infrastructure was helpful in energy management for the city. The authors aim to extend the infrastructure by adding the big data analytics module for analyzing and making use of the historical data being collected by the infrastructure. InterSCity is one of the open-source

and microservice-based middleware available to the community to improve and contribute to research [53]. InterSCity is the middleware for IoT for applications in the domain of smart cities with flexible, extensible, scalable, and loosely coupled architecture. It spans the entire smart city domain and does not target a specific application area. An application is built using the proposed middleware to validate the scalability, performance, and usage of the middleware. Two main drawbacks of the proposed paradigm are the increased operational complexity and the inefficient handling of large volumes of data.

3.1.5. Other Middleware

There are some proposals which did not mention any underlying architecture. These are discussed in this section.

An architecture is proposed to deal with the energy efficiency of IoT resources [54]. The authors have used the mechanism of setting the devices to sleep mode, depending on the requirements and quality of service required by the applications and the current battery level of the devices. They also devised a mechanism to re-provision the allocated cloud resources when the corresponding IoT devices are in sleep mode. The proposed architecture makes use of the sleep scheduling method, which saves the energy of the IoT resources and makes the utilization of cloud resources efficient. It is useful in many scenarios because it is an architecture and not a method. Five volunteer individuals were equipped with three sensors for the experimentation setup. These were blood pressure monitor (BP), heart rate monitor (HR), and respiratory rate monitor (RR) sensors. The mobile phones of the individuals were used as energy-efficient gateways, and these were used to control the sleep interval of the sensors. BP and HR are the periodic sensors, whilst RR is a trigger-based sensor. A mobile phone working as eGN is used to send the data received from the sensors to the Cloud for further processing.

A solution using ontology and JSON-LD to annotate connectivity [55], security, and privacy properties of IoT devices. The authors also developed an application-level protocol wrapper for communication consistency, integrity, and secrecy for low-cost devices with cheap microcontroller units (MCUs).

The newly created ontology is richer than the ontologies in the past. It has the concepts of annotating the privacy, security, and supported protocols as non-functional properties of things and their services. JSON-LD's use helps to annotate things using the newly created ontology. The current protocol wrapper is used and tested on devices with two MCUs only and may not work smoothly on the other devices with different MCUs, which is one of the solution's limitations.

3.2. Critical Evaluation of Different Middleware Architectures

In the literature review, we studied the middleware solutions for IoT-based environments. We learnt that, although there are a number of these proposals, these lack one or the other aspect of the application requirements for the smart cities domain and thus do not provide a comprehensive solution. These either have limitations and weaknesses or were not tested thoroughly enough to prove the stated claims. For instance, in [56], the authors evaluated four middleware proposals. These are OpenIoT, CHOReOS, LinkSmart, and UBIWARE. The authors evaluated these for autonomous services, scalability of the service registration, discovery and composition of services, interoperability, and heterogeneity of IoT devices. The evaluation results show that all the middleware follow a semi-automated registration process and semi-autonomous discovery component. None of the evaluated middleware provides the automated service composition. Therefore, all of these are deficient for certain functionalities, and further research is required.

The previously proposed frameworks reviewed in this work, along with their domain areas, architectures, limitations and evaluation parameters with results, are listed in Table 2 and show only studies that provided the evaluation parameters and testing.

Table 2. Critical evaluation of middleware for smart city applications.

Middleware	Architecture	Domain	Limitations/Weaknesses	Evaluation Parameters
SmartCityWare [45]	Service-Oriented	Smart Cities/Generic	Did not implement all the listed services.	Communication, Performance Time Required: Local calls = 500 ms Remote calls = 3000 ms Service lookup = 1300 ms–2400 m
MinT [25]	Component-Based	Smart Cities/Generic	CPU resources wastage if number of requests are smaller than the throughput of the thread pool. The creation of threads also increases memory use. Fewer platforms/operating systems support.	Average throughput per second: Throughput = 8900 requests/s
InterSCity [53]	Microservice-Based	Smart Cities/Generic	Inefficient data handling by multiple databases. Increased operational complexity due to decentralized databases.	Performance, Scalability [With 6 Resource Adaptors 1546 requests/s]: For Performance: [<1 s for 350 parallel clients]
ABC&S [41]	Service-Oriented	Smart Cities/Car Parking	No criteria defined for the best parking lot.	Response Performance: Average Response Rate < 1 s, i.e., in ms
CoTWare [43]	Service-Oriented	Smart Cities/Generic	Challenges addressed are unclear. Must have implemented all the stated services.	Communication, Performance Time Required (10 Calls mean values): Local calls = 500 ms Remote calls = 1700 ms Service lookup = 50 ms–1200 ms
SMArc [32]	Distributed	Smart Cities/Energy Management	Lack of GUI. The results are highly dependent on nature of device and implementational scenarios.	Service Registration vs. time in ms, Service requests completion vs. time: Average of service registration = 453.4 ms Average of Simple services = 561.2 ms Average of Composed service = 661.1 ms
FogFlow [36]	Distributed	Smart Cities/Generic	Low throughput of brokers with increasing subscribers. Low throughput of queries and response times.	Throughput and response time/message: For geoscopic-based queries: Average Response Rate = 1000 ms (Approx.) For ID and topic-based queries: Average Response Rate = 100 ms

Table 2. Cont.

Middleware	Architecture	Domain	Limitations/Weaknesses	Evaluation Parameters
DAQ-Middleware [26]	Component-Based	Smart Cities/Generic	In the case of uniform interfaced data sources, the parallel data acquisition algorithm may not be very useful. The efficiency of DAQ, along with the configuration tool, is quite low at the start.	Time required to complete a data acquisition: By DAQ = 387 ms By Serial = 1979 ms
Rimware [42]	Service-Oriented	Smart Cities/Generic	No testing for scalability is provided.	Security and Authentication initialization time: Security initialization time: 135 ms Authentication initialization time: 135 ms
Service-Oriented Middleware [44]	Service-Oriented	Smart Cities/Smart Grid	The middleware should have been applied to an actual smart grid environment to prove its validity.	The best service quality and the metric used is Mean Opinion Score: MOS = 4.3 (N = 10) MOS = 4.1 (N = 20) MOS = 4 (N = 30) MOS = 3.95 (N = 40)
Soul [35]	Distributed	Smart Cities/Generic	Lack of testing results for real-time streaming big data.	Scalability: Messages = 10,000/s
End-to-End IoT Security Middleware [28]	Component-Based	Smart Cities/Generic	Due to the use of the session resumption, trust state is required to be maintained in the network amongst the IoT devices. The short-lived reputation or trust state must be supported to avoid maintenance of trust state for longer periods and dealing with the transient devices.	Time for Security Association and Session Resumption time: Security Resumption: 40 ms Security Association: 80 ms
DDS [34]	Distributed	Smart Cities/Generic	DDS is not implemented with an adequate number of IoT middleware. Should have been tested for more detailed performance evaluation.	Performance, Scalability For Synchronous Collection: Messages = 11,000/s (1 node) Messages = 25,000/s (3 node)
Internet of Things: a N Interoperable IoT Platform [7]	Not Specified	Smart Building	No prototype is developed to validate the proposed framework.	No evaluation parameters are provided.
MiSCi [40]	Service-Oriented	Smart Cities/Generic	Not validated for real use cases, but only tested using the simulations.	Timeliness of monitoring, analysis, and execution of required action in different scenarios.
SGeol [23]	Component-Based	Smart Cities/Generic	Validation is done using simulations and virtual machines.	Number of concurrent handled requests in one minute. Min = 51,887, Max = 54,414, Avg = 53,661.25 (for a single SGeol Core instance)

Table 2. Cont.

Middleware	Architecture	Domain	Limitations/Weaknesses	Evaluation Parameters
An agent-based middleware framework based on distributed CPS [37]	Distributed	Smart Cities/Generic	Validation is done using the simulations, and middleware is not tested by developing any applications.	Average resource utilization vs. queries and time. Storage utilization vs. queries and time. Downtime vs. queries and time. Response time vs. queries.

All of the previous work has addressed a few of the challenges in the domain of middleware. Most middleware has addressed heterogeneity, flexibility, scalability, extensibility, and data acquisition. Therefore, these challenges have more importance than the rest, which are the domain's core. For a comprehensive solution in the area, it is a must to address all of these challenges along with the others. Still, it is obvious that missing any of these attributes will have a significant impact on the effectiveness of the solution. Table 3 lists the challenges addressed by each of the proposed frameworks. The crux of the study with different aspects and benchmarks is listed in Table 4.

Table 3. Challenges that each middleware addresses.

Middleware	Data Acquisition	Scalability	Heterogeneity	Flexibility	Extensibility	Security
ABC&S [41]		X				
Device Nimbus [24]		X		X	X	
InterSCity [53]		X		X	X	
GAMBAS [27]	X					
Civitas [33]	X		X			X
MinT [25]		X	X			
Rimware [42]						X
FogFlow [36]		X		X		
CoTWare [43]		X	X	X	X	X
SMArc [32]		X	X			
Service Oriented Middleware [44]			X			
SmartCityWare [45]		X	X	X	X	X
Distributed Data Service [34]	X					
DAQ-Middleware [26]	X					
Data Processing Middleware [46]			X			
AUSOM [50]						
AndroAec, D., et al. [55]						X
Brundu, F. G., et al. [52]			X			
Soul [35]		X				
SAVI-IoT [47]			X			X

Table 3. Cont.

Middleware	Data Acquisition	Scalability	Heterogeneity	Flexibility	Extensibility	Security
TinyCO [51]						
Kaur, N. and S. K. Sood., et al. [54]						
Zgheib, R., et al. [31]						
LISA [48]						
Mukherjee, B., et al. [28]						X
Abreu, D. P., et al. [30]		X				
Mint-I [29]		X	X			
SEEMPubS [57]		X	X			
Patti, E., et al. [49]			X			
MASSIF [39]		X			X	
A new Interoperable IoT Platform [7]			X			
MiSCi [40]	X	X		X	X	
SGeol [23]			X		X	X
GMSCA [Proposed Middleware]	X	X	X	X	X	X
Count	4	13	12	5	5	7

Table 4. Extracted aspects and benchmarks.

Aspect/Benchmark	Category	Value
Famous Architecture	Aspect	Service-Oriented Architecture
Mostly Addressed Challenges	Aspect	Heterogeneity, Scalability, Flexibility, Data Acquisition, Extensibility, and Security
Evaluation Parameters	Aspect	Performance: Throughput, Service Requests
Throughput Messages/s	Benchmark	8900/s
Requests Completion [Server/Cloud-Based]	Benchmark	1700 ms
Requests Completion [LAN-Based]	Benchmark	500 ms
Data Acquisition	Benchmark	387 ms

3.3. Proposed Architecture for GMSCA Middleware

As per the discussion in the above section, it is evident that there is a need for a comprehensive solution that addresses all the core identified challenges with a system which is flexible enough to incorporate the services that would emerge at later stages, along with the flexibility of changing the services overtime upon need. This section presents a complete framework for the end-to-end architecture of IoT-based applications and the proposal for middleware for IoT-based applications. It contains all the necessary components and advanced components, such as data analytics and Artificial Intelligence (AI) components, to support different smart cities applications.

Both data analytics and AI components are necessary, given the massive amounts of data being collected nowadays. Smart cities produce data quickly due to a large number of data-producing points at different intervals. Big data analysis is, therefore, a must to process such huge amounts of data. AI components could then be augmented with big data analytics, and this data could be used for useful purposes such as future predictions

(classification), identification of different data patterns (association rule mining), and grouping of the data using clustering.

GMSCA contains these components to leverage the benefits of advanced technologies. Data flow is then shown in the overall system and how different system components interact to serve the purpose and fulfil the application requests. A hypothetical example further explains a real-life scenario of initiating and completing an application request.

Based on the previous literature, the basic architecture of IoT-based applications is shown in Figure 1. These are five necessary layers to develop applications based on the IoT. All the layers in the architecture are involved in two-way communication with their immediate upper or lower layers.

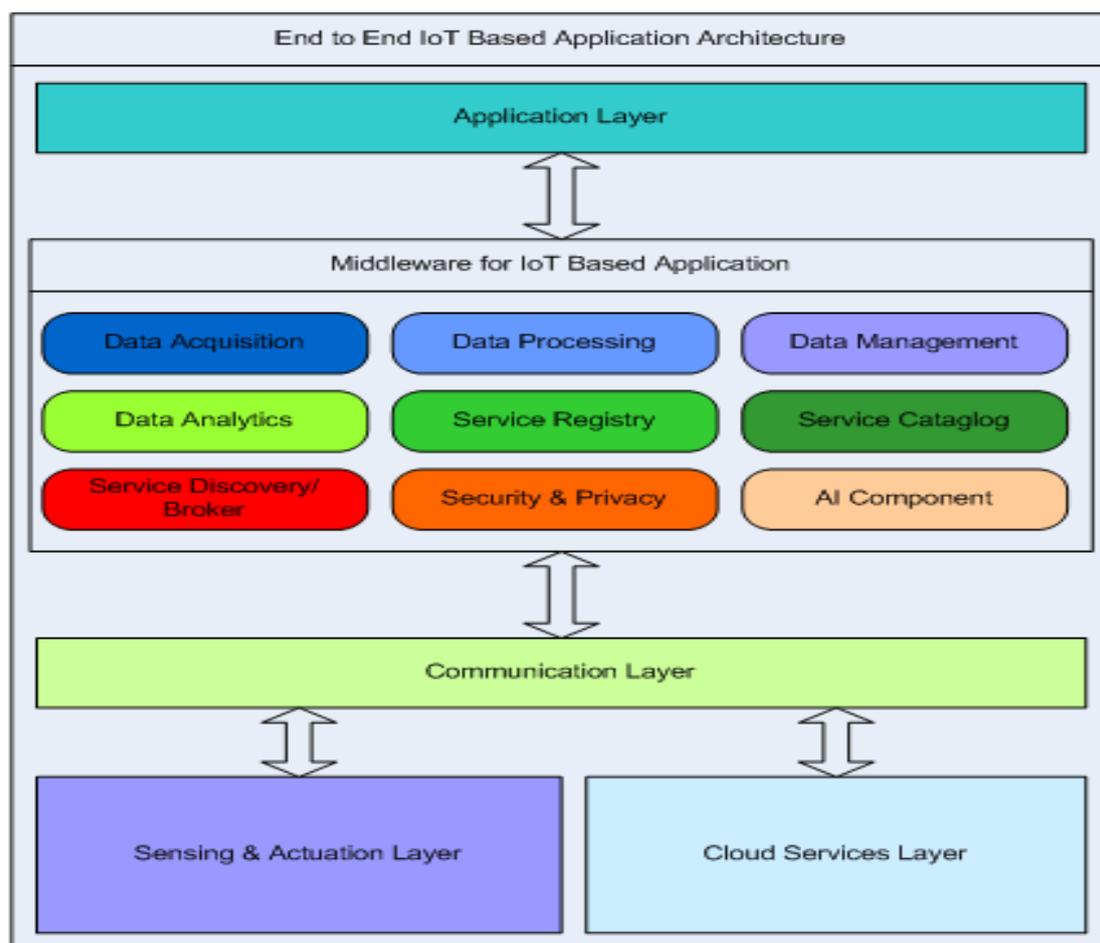


Figure 1. IoT application architecture.

The sensing and actuation layer sits at the bottom of GMSCA architecture and helps the data acquisition process by sensing the environment using different sensors, such as camera sensors, humidity, or temperature sensors. Data from the sensing layer is then transferred using the communication layer, which resides on top of the sensing and actuation layer. The communication layer communicates with middleware and the cloud services for data, either serving the application layer via middleware or storing the data in Cloud. The middleware layer is the core of the IoT system, as it may contain different components to perform various operations required for the development of applications for IoT-based systems. The Cloud layer is usually responsible for storage. Different types of databases could be used for storage purposes, such as relational databases and NoSQL databases, among others. The application layer contains all the user applications, such as mobile applications and web applications. These applications use the components in the

middleware to perform different operations such as acquiring data using API calls, and processing and storing data.

The following sections provide an overview of the GMSCA middleware components and how these components work together to address the challenges of the smart city application domain. The middleware layer is the central area of interest as it contains all the components of the GMSCA, and Figure 1 shows the component-level view of this layer. The components include data acquisition, data management, data processing, data analytics, service registration, services catalog, service discovery/broker, security & privacy, and artificial intelligence components. A short description of these components is given below.

The data acquisition component, also known as the data collector component, is responsible for data acquisition or actuation for the data originating from the sensing and actuation layer. The data is then processed using the data processing component and then managed/stored using the data management component. The data analytic component is responsible for data visualization, data aggregation, and summarization.

The devices and services are registered using the services registry component and recorded in the services catalog. The available services are then discovered using the service broker/discovery component based on the application queries and services requirements. Data security and privacy are provided using the security and privacy component by using different strategies.

The challenges of data acquisition, scalability, heterogeneity, flexibility, extensibility, data analytics, and security are handled through different components. For instance, the data collector component is used to address data acquisition challenges. This component accepts the data from the gateways and sends it to the server/cloud using the middleware's API calls. NoSQL database is used for storage purposes to deal with the heterogeneity of the devices' data and data from other sources. NoSQL is a schema-free database (i.e., it does not need any configured schema) that can incorporate any schemas. As a result, with the introduction of any new device with a different format, schemas are created for the device without affecting the previous schema and services. The security component is implemented in the middleware to enforce the security mechanism for data and service usage. In the security component, a role-based authorization is used to allocate separate rights to application users, developers, and data administrators. Privacy is implemented using data encapsulation and data hiding. Instead of revealing the actual data (wherever required), the processed data is provided to application developers.

The middleware is based on service-oriented architecture to deal with scalability, flexibility, and extensibility challenges. As a result, all services can be implemented and plugged into the middleware without affecting the other services. It makes the individual services more manageable, scalable, flexible, and extensible by introducing more and more services upon the new emerging requirements. Scalability can also be achieved using replication, and it can also be used to improve the system's performance. On top of these advantages, the GMSCA integrates the services provided by different vendors in the city and this, in turn, carries the advantages of microservices-based architecture and offloads the processing work for those services to the third-party vendors' servers. GMSCA application developers use the data which is queried through the web services provided by the aforementioned third-party vendors; still, the middleware encapsulates the web service calls from vendors, and it maps the API calls for developers to native GMSCA's API calls. As a result, the developers' query data appears from the GMSCA's internal data repository.

The middleware implements two components for predictive data analytics and data visualization: the data analytics component and the AI component. The data analytics component supports the functions that help with data visualization and aggregation, whilst the AI component implements predictive analytics through machine-learning and other data-mining techniques.

Figure 2 shows the overview of data flow from the sensing and actuation layer towards the application layer on initiating a request by the application users. It shows how different layers and components communicate with each other to make the overall system work.

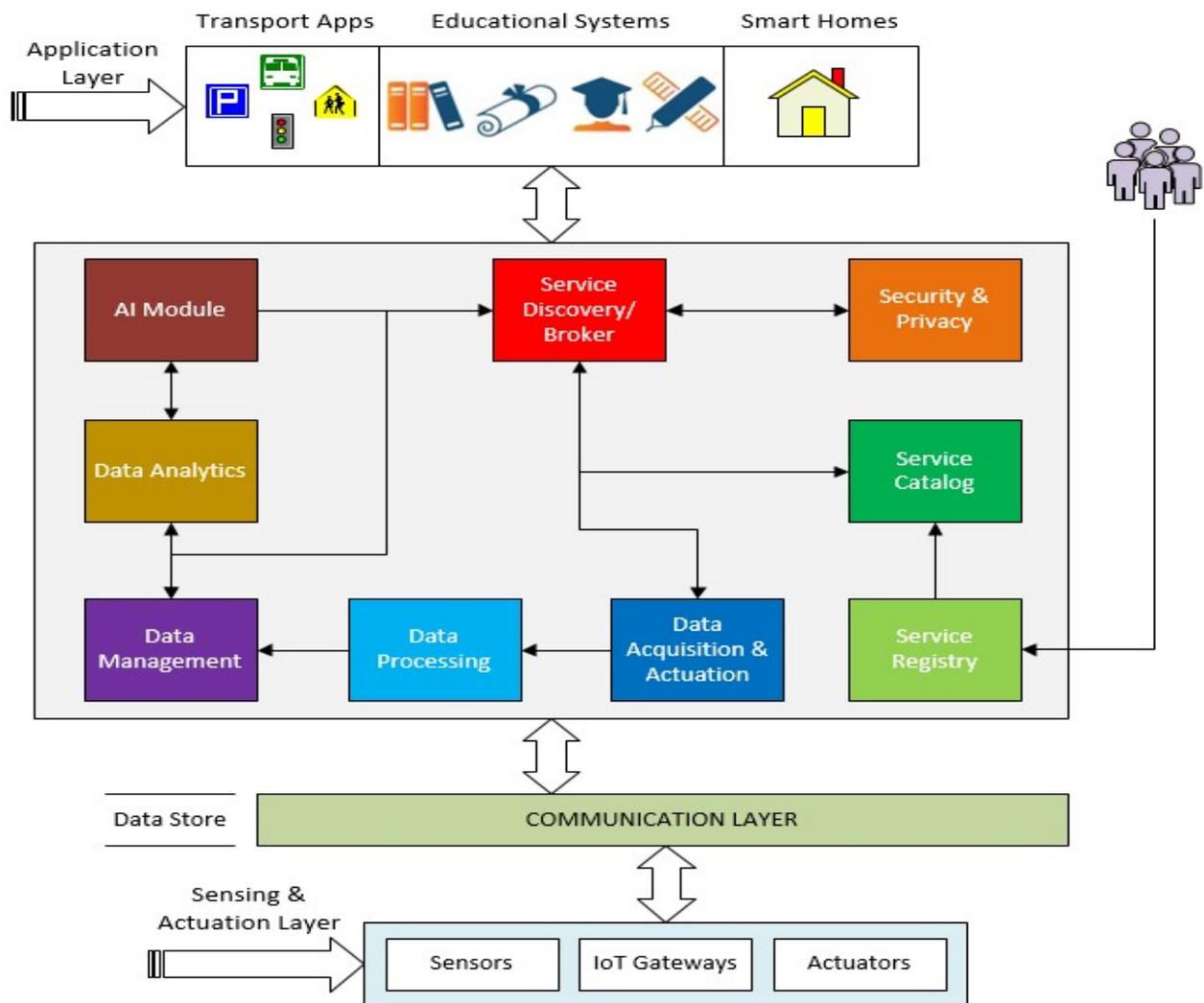


Figure 2. Proposed middleware architecture and data flow in middleware components.

The following example demonstrates how the interaction of the different components and services will fulfil a user request in the proposed layered architecture. For instance, if a user wants to know the temperature of a specific location, the user sends a request to the smart city application using a mobile phone. The application request is then processed by the service broker component in the middleware to find the appropriate service. The system checks the user's authorization, and if the user appears to be authorized to use this service, environmental data is sensed using the data acquisition component. Temperatures from different corresponding data points are gathered and passed to the data analytics component to be processed and aggregated. The final average temperature is then forwarded to the user who requested the service.

4. Implementation Details of GMSCA

The proposed middleware is implemented using a set of technologies that include Python, Flask, Spark ML, Pymongo, Spark, and MongoDB. The front-end application for the access and description of API and services is implemented using PHP. The core middleware API is written using Python and a Flask framework [58]. Flask is a microframework for Python based on Werkzeug and Jinja 2. Flask helps in RESTful request dispatching.

Spark is a high-speed, in-memory data processing engine [59] which is used as the processing engine by GMSCA to develop the predictive analytics machine-learning models using the Machine Learning (ML) library [60]. Spark can run on the cluster and bring

scalability through horizontal scaling. Spark was chosen in the present study because iterative processing of the data is required, and this is facilitated by Spark's capability of in-memory data processing and caching.

MongoDB is used for data storage due to its useful features, such as a dynamic schema and rapidly changing structured, semi-structured, and unstructured data [61]. NoSQL should be the choice in a smart city application because of heterogeneous data sources and variant data structure. Moreover, NoSQL allows flexibility in changing the schema over time without breaking or affecting the dependent components. A Pymongo wrapper is used in Python for communication, data persistence, and retrieval from the MongoDB. Further details on the usage of the different components are discussed below.

The core API of GMSCA contains all the API calls that use different services and components. It sends the results to the application developers in JavaScript Object Notation (JSON) format [62]. JSON is a standard format for data interchange on the Internet. As a result, application developers can avoid dealing with heterogeneous formats and can use and interpret one single format, making integration and working with data and services easier. The core API also receives the API calls from developers and transforms them into third-party application service calls when required.

The core API also contains the calls that use the data collector component for data collection from the IoT devices and is responsible for data actuation using IoT gateways. Application developers communicate with the components of the system, including data analytics and AI components, using core API calls.

The description of services and their corresponding calls are listed on the front-end web application, which is built using PHP. Third-party vendors who can provide the data for the benefit of the community can create their services using the aforementioned web application. Their direct web service calls that are responsible for data retrieval are protected and transformed into a GMSCA native application call, which is then made available to application developers to develop various applications.

Spark ML is used for the development of various machine-learning models for the purpose of prediction and data visualization, summarization, and aggregation. The ML library [60] includes the functions that contribute to the development and implementation of various machine-learning algorithms for the creation of various predictive models and other statistical functions.

MongoDB is a NoSQL database [61] used for data storage purposes in the system. It brings many advantages to the system, such as a dynamic schema and the capability of semi-structured data storage. It stores the records as documents and returns the results as JSON-formatted objects, which are easier to interpret and use by most applications and tools. Pymongo is used in the Core API to communicate with the database.

5. Applications Using GMSCA

Three minimum viable applications are developed using GMSCA to demonstrate its feasibility and the functioning of different components in the system. Details of these applications are presented below.

5.1. Web-Based GUI for GMSCA

This web-based application is developed using the GMSCA API. A snapshot of the interface is shown in Figure 3. The interface was developed to provide a communication platform for the following stakeholders: third-party vendors, developers, administrators, and city residents.

- To use the underlying services by the third-party vendors, they need to sign up, as shown in Figure 4. Following successful registration, they register their services as service calls designating their data to be fetched by the system. These service calls to vendor services are then mapped to a GMSCA native service call and stored in the database. Vendor-specified service calls are protected, and developers only see the GMSCA's native API calls in their applications. These calls are then received by the

GMSCA system, which maps the calls to the required actual service call and pushes the fetched data from the called service to the caller.

- The application developers can use the service platform to review the available services and the data descriptions. A sample of available services is shown in Figure 5. After selecting the right service, the developers can use the service calls in their applications and use the fetched data to provide the relevant functionality in a specific application. Various services might need authorization permissions, which the GMSCA administration provides in coordination with the vendors. An example of the available services could be ticket registration with travel companies. In this case, availability and ticket information may be fetched and used without authorization, but the tickets could only be booked using the authorization.
- The GMSCA administration uses the platform to approve the vendor's posted services. These services are properly reviewed under certain service contracts before making available to application developers.
- Smart city residents can use the platform to explore services and discover different applications developed using middleware. This helps them find everything built for them using GMSCA and contributes to a better quality of life in the city.

Our Services

[Add New](#)

Title:	Bus Ticketing Service
Description:	Skyways Bus Ticketing Service
URL:	http://www.gmsca.com/api/skyways/getticketinfo
Data Description:	Now info for all our tickets is available online
Documentation:	Get all the tickets info regarding pricing and routes, destinations and dates.

Title:	Bus Ticketing Service
Description:	Faisal Movers Ticketing Service
URL:	http://www.gmsca.com/api/faisal movers/ticketinfo
Data Description:	Will provide ticket info for passenger
Documentation:	Complete ticket info and pricing for our valued passengers

Copyright © 2018 GMSCA . All Rights Reserved. [Privacy Policy](#)

Figure 3. Third-party vendor's services page.

Sign Up

Company:

Phone:

Mobile:

Email:

Website:

Address:

Contact Details :

First Name:

Last Name:

Designation:

Phone:

Mobile:

Email:

Company Description:

Username:

Password:

Already have an account? [Click Here](#) to login.

Figure 4. Web-Based GUI for company sign-up and company services registration pages.

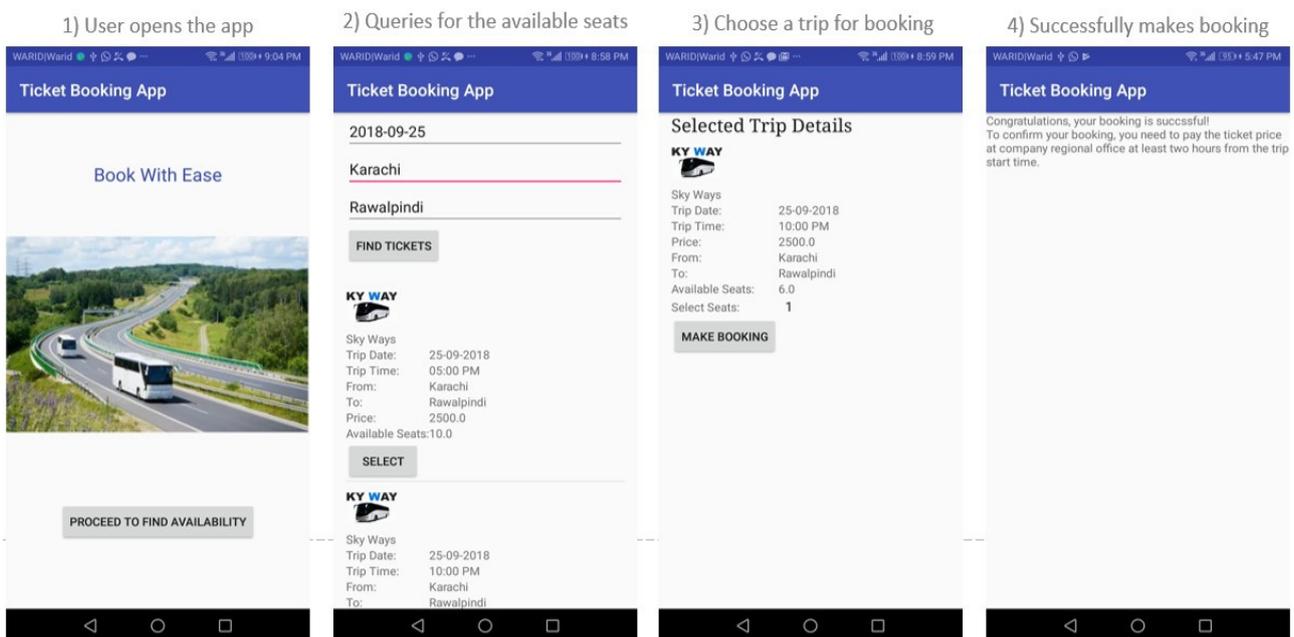


Figure 5. Flow in ticket-booking mobile application.

5.2. Bus-Ticket-Booking Mobile Application

A bus-ticket-booking mobile application is developed using Android Studio [63]. This application is developed to show the usability of GMSCA, where vendors can post the services; smart city developers use the resulting data to build useful applications for city residents.

To develop this application, mock data, provided services, and vendors were used. These mock objects resemble the actual data (although fields and data may change) to show the suitability of GMSCA for such real-life scenarios.

To this end, a bus ticket booking mobile application uses the data provided by different bus service companies. The service calls fetch information related to trip details, company information and ticket availability.

Applications users can search through these by entering criteria such as trip start location, destination, and date. Depending on these criteria, a service call is created on run time, and the resulting data is retrieved and displayed to the user. The user can then choose a ticket to book depending on its suitability and is then shown a screen where the number of tickets to purchase can be chosen. Tickets can then be booked by using another API call to the respective travel company or vendor. The user is then notified of the booking's success or failure. Snapshots of the user-search-screen-retrieved ticket info, booking page, and booking-success screens are shown in Figure 5.

5.3. Data Acquisition & Actuation Application

GMSCA can help acquire and store the IoT devices' data using its API calls and gateways. API calls are made by IoT gateways to push or pull data from the system. Data is pushed in case of data acquisition or collection and pulled in case of actuation.

GMSCA have been tested for the performance parameter. The data acquisition application is used for communication with the result of data acquisition from IoT devices by the GMSCA hosting server. The test results for twenty data acquisition operations are shown in Figure 6. The average time for data acquisition operations, along with the upload to the server using middleware API calls and data saving to MongoDB, is 802 milliseconds.

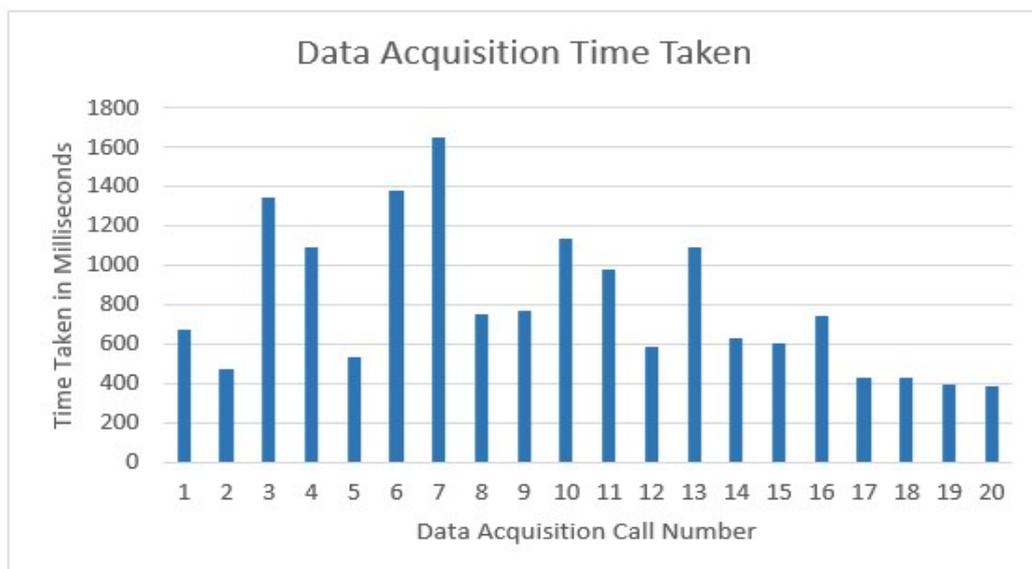


Figure 6. Results of data acquisition using GMSCA API calls.

For implementation purposes, a desktop application is developed using the Microsoft.net framework and used as the gateway for IoT devices. Arduino UNO is used for communication and data retrieval from the DHT sensor. The DHT sensor provides the data for temperature and humidity. This data is transferred using the Arduino serial interface to a desktop-based application (acting as a gateway in our present case), which uploads the

data to the GMSCA data persistence repository based on MongoDB. For this experiment, we used the VM running Centos Linux 7 with 12 GB of memory and four processor cores. API calls are sent using Ngrok [64], a platform that provides secure tunnels to localhost. All our requests were then routed through Internet using API calls.

For the implementation of the actuation application, API is hosted on Intel's core I7 computer with 16 GB of memory running on Windows 10. Ngrok is used to send all calls to API to introduce the delay caused by internet routing. A push button is used to send the events to the system, which sends back responses to switch on/off the LED, showing the feasibility of actuation applications using GMSCA's API calls. The experiment results are shown in Figure 7.

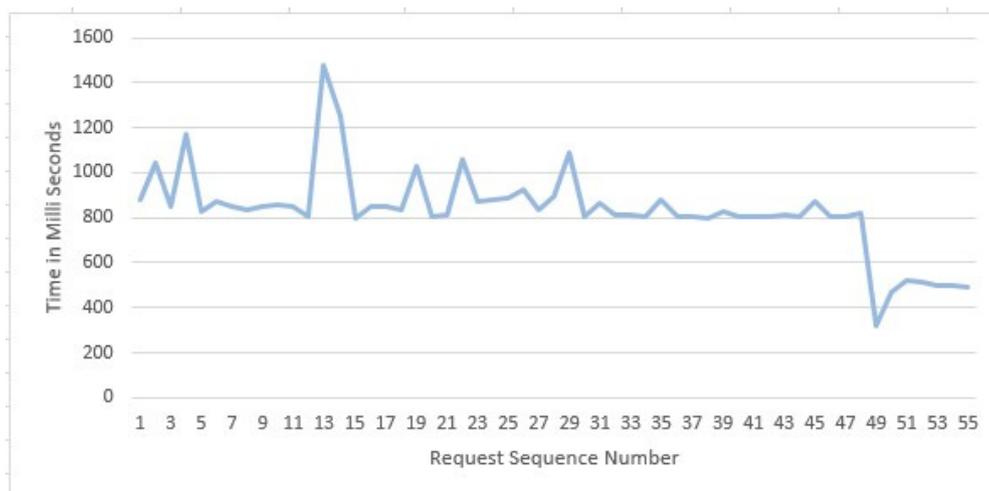


Figure 7. Request and response graph for API calls.

The average time taken from request generation on the client side to the response receipt by the same client was observed to be 831 ms. Snapshots of the hardware devices used in the experiment, i.e., Arduino UNO, DHT sensor, push button, and LED, are shown in Figure 8.

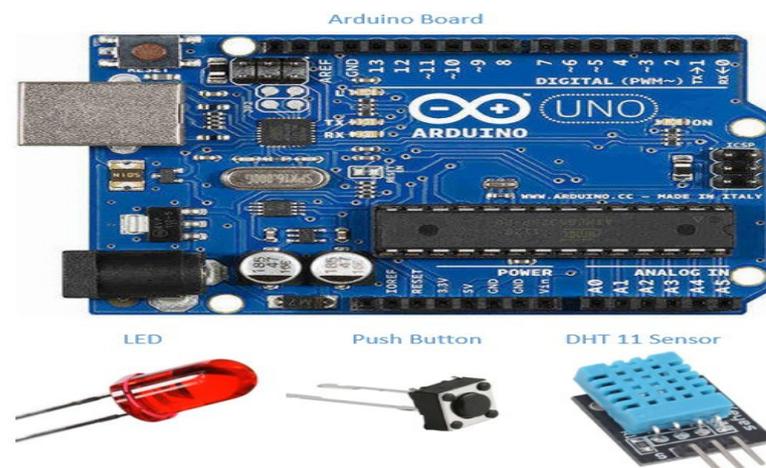


Figure 8. Hardware used in data acquisition.

Under the same setup, the experiment was repeated to record the time necessary to send the API calls to the proposed middleware (just as in phase 1 of this application). In this scenario, the time taken did not include data saving to MongoDB. API calls, however, are sent using ngrok. The results of this experiment are shown in Figure 9. The average time from request generation to receipt of the request at the server is around 650 ms.

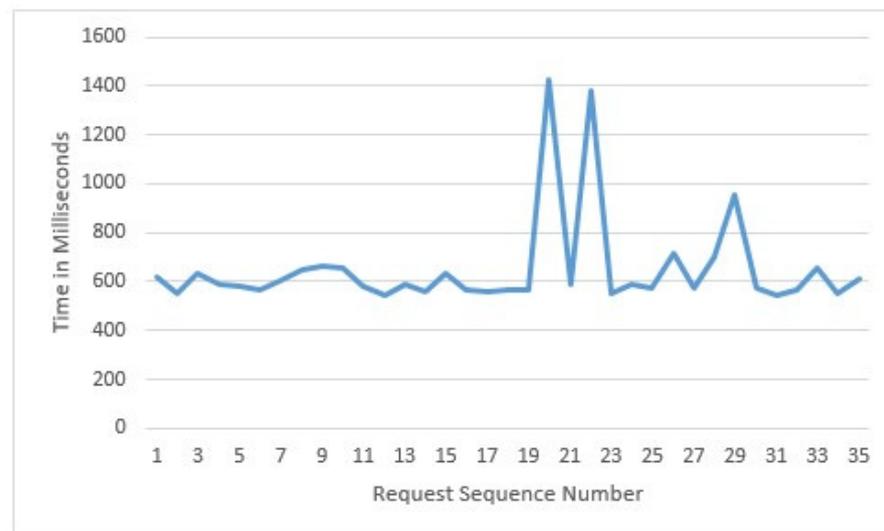


Figure 9. API Call receipt times from client to server.

6. Load Testing

The performance of the proposed middleware was evaluated and compared with similar architectures through load testing using Locust [65]. This tool has a simple interface for inputting the number of concurrent users and hatching time. Hatching time was set to 500 ms for all experiments with different numbers of concurrent users. Locust has the browser-based visualization and capability of displaying tabular results and drawing charts for different parameters. Tabular results are shown in Figure 10. Load testing was repeated for different numbers of users, and the median, average, and minimum response times were recorded for each test. The results are shown in Table 5.

# requests	# fails	Median (ms)	Average (ms)	Min (ms)
55298	11	1700	1771	18.98503303527832
55298	11	1700	1771	18.98503303527832

Figure 10. Average response time by concurrent users.

Table 5. Load-testing results of concurrent users.

Concurrent Clients	Total Requests	Times (in Milliseconds)			Failed Requests %
		Median	Min	Average	
250	52,131	1.1	0.018	1.114	0.01
300	50,018	1.2	0.035	1.246	0.02
400	55,880	1.2	0.038	1.302	0.02
500	53,526	1.3	0.010	1.421	0.02
600	55,298	1.7	0.018	1.771	0.02
700	52,467	2.0	0.019	1.936	0.02

Figure 11 shows request/response times as a function of the number of concurrent users. It indicates that the average and median times are almost overlapping, which shows the linearity in the pattern of request/response times

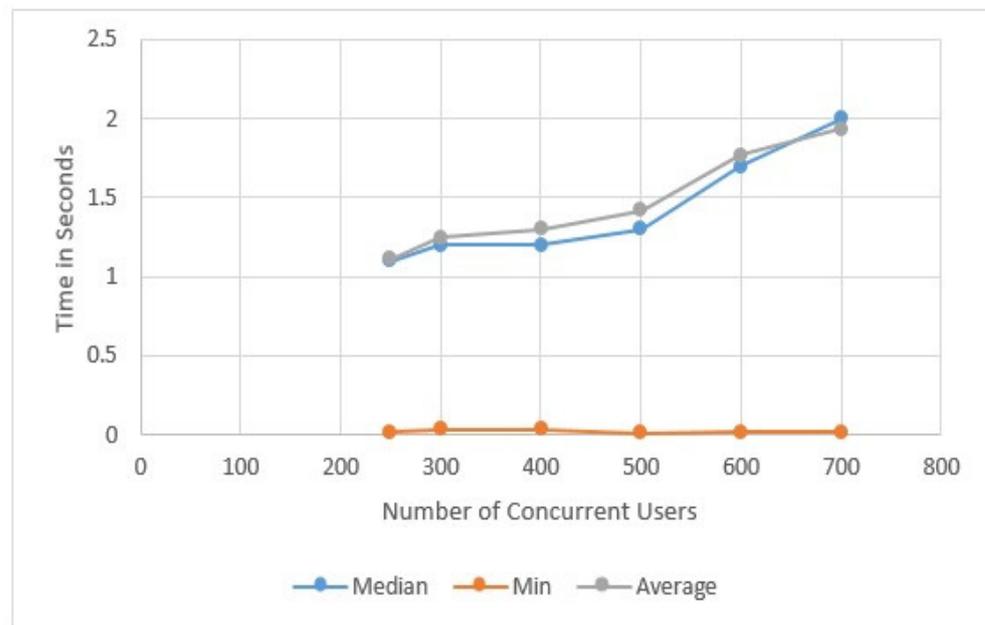


Figure 11. Load-testing response time for concurrent users.

The results show that the response time starts to display a latency of 2 s from around 700 users. Below 700 concurrent users, the average and median response times are below 2 s, which is reasonably satisfactory for applications that do not have stringent real-time response requirements. These response times could substantially be lowered through load balancing and replication strategies. It is also evident from the results that the x percentage of failed requests is quite low for a number of concurrent users, up to 700:0.02% for 700 concurrent users and more than 50,000 requests.

The results of the request completion of GMSCA were compared with SmartCityWare and CotWare, shown in Figure 12. Moreover, load-testing results of GMSCA were compared with those of the InterSCity middleware shown in Figure 13. In both cases, the request and response time of GMSCA is far lower than other middleware used for comparison.

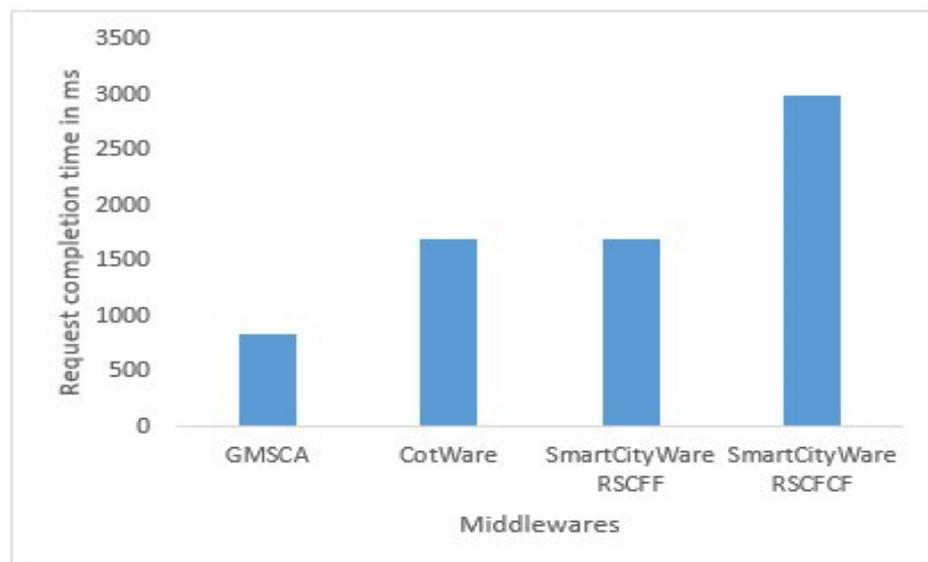


Figure 12. Comparison of GMSCA request completion time with other middleware.

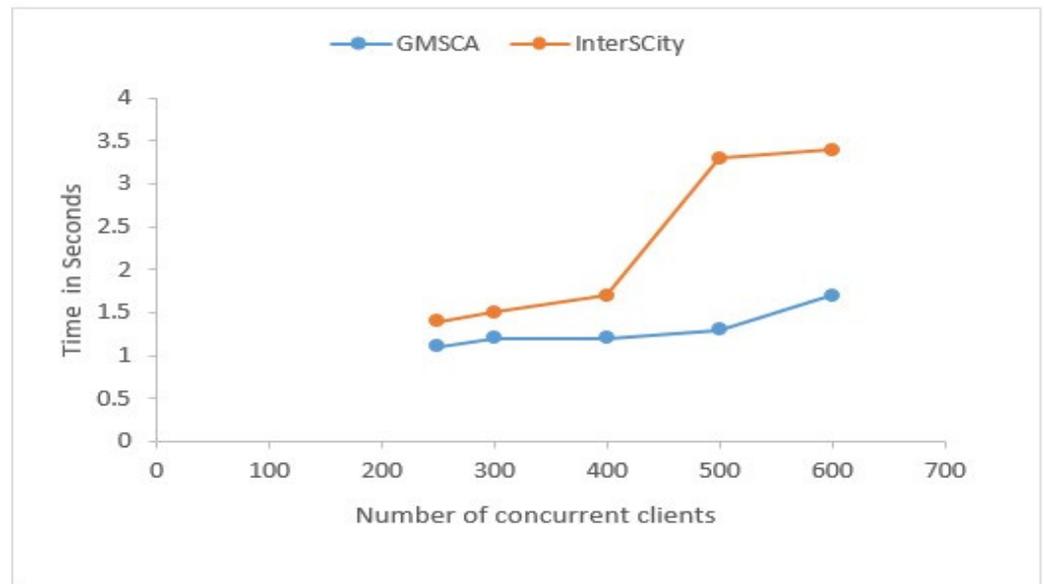


Figure 13. Comparison of load-testing results of GMSCA with InterSCity middleware.

7. Conclusions and Future Work

The proliferation of the Internet of Things (IoT) in several application domains necessitates the establishment of a well-defined infrastructure of systems that can support the development of applications, as well as provide services for the abstraction and management of data pertaining to connected devices. Over the past few years, middleware for the IoT has emerged as an increasingly important component because it has been identified as the system that is able to deliver the essential infrastructure of services. The aim of this paper was to review and synthesize the relevant literature to identify and discuss the challenges of existing architectures of IoT middleware systems. A critical analysis was performed to highlight their strengths and weaknesses. The comprehensive analysis identified that none of the existing middleware systems has addressed all the key challenges; hence, choosing one as a reference architecture was difficult.

We proposed and implemented a generic reference architecture, GMSCA, by combining the information landscape of existing IoT middleware systems with big data applications to achieve the required level of services supporting sustainable cities. GMSCA is a generic middleware architecture proposed for the development of smart city applications. It addresses the core challenges, including data acquisition, scalability, flexibility, heterogeneity, extensibility, and security. A basic implementation of advanced data analytics and artificial intelligence components is provided and plugged into GMSCA by implementing three applications. One is web-based, the second for mobile devices, and the last deals with data acquisition for the actuation of connected IoT devices. The results show a satisfactory performance of the GMSCA and the middleware could be used in IoT application development for future cities.

The performance of the GMSCA was compared to that of other middleware for request completion times and load testing for concurrent users. The results showed the advantage of GMSCA in terms of performance over other middleware. Data analytics components and AI components are developed and plugged into the middleware, but their testing by developing applications based on these components is yet to be completed. A complete description of these components and test results is planned. The other challenges, such as resilience, interoperability, and QoS, will also be addressed in future implementations to build a comprehensive middleware system for various IoT-based applications, especially applications targeting the smart city domain.

Furthermore, the study only reviews the most commonly used IoT middleware designed to develop IoT-based big data applications. To further validate the performance of

the proposed middleware, it might be helpful to compare it with state-of-the-art middleware architectures. In future work, we plan to conduct an empirical analysis and compare the performance of the proposed architecture with state-of-the-art middleware architectures to further evaluate its suitability for designing smart city applications.

Author Contributions: Data curation, Z.A., W.A. and A.M.; Methodology, Z.A., S.K. and A.M. Writing—original draft, Z.A., A.M., S.K., S.S.U., J.I. and S.H.; Writing—review & editing, W.A., S.H., S.K. and A.M. Supervision, A.M. and S.K. Funding acquisition, W.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Taif University, Taif, Saudi Arabia, grant number [TURSP-2020/107].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank Researchers Supporting Project number (TURSP-2020/107), Taif University, Taif, Saudi Arabia.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Li, S.; Xu, L.D.; Zhao, S. The internet of things: A survey. *Inf. Syst. Front.* **2015**, *17*, 243–259. [\[CrossRef\]](#)
- Kiran, M.; Wynn, M.G. The Internet of Things in the Corporate Environment: Cross-Industry Perspectives and Implementation Issues. In *Handbook of Research on Digital Transformation, Industry Use Cases, and the Impact of Disruptive Technologies*; IGI Global: Hershey, PA, USA, 2022; pp. 132–148.
- Petrolo, R.; Loscri, V.; Mitton, N. Towards a smart city based on Cloud of things, a survey on the smart city vision and paradigms. *Trans. Emerg. Telecommun. Technol.* **2017**, *28*, e2931. [\[CrossRef\]](#)
- Santana, E.F.Z.; Chaves, A.P.; Gerosa, M.A.; Kon, F.; Milojicic, D.S. Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 78. [\[CrossRef\]](#)
- Chelloug, S.A.; El-Zawawy, M.A. Middleware for Internet of Things: Survey and Challenges. *Intell. Autom. Soft Comput.* **2017**, *24*, 309–318. [\[CrossRef\]](#)
- Adepu, A. IoT-New Trends in Middleware Technologies. *Int. J. Adv. Res. Comput. Sci. Manag. Stud.* **2017**, *5*, 48–58.
- Abdelouahid, R.A.; Debauche, O.; Marzak, A. Internet of Things: A new Interoperable IoT Platform. Application to a Smart Building. *Procedia Comput. Sci.* **2021**, *191*, 511–517. [\[CrossRef\]](#)
- Gazis, A.; Katsiri, E. Middleware 101. *Commun. ACM* **2022**, *65*, 38–42. [\[CrossRef\]](#)
- Farahzadi, A.; Shams, P.; Rezazadeh, J.; Farahbakhsh, R. Middleware Technologies for Cloud of Things—a survey. *Digit. Commun. Netw.* **2017**, *4*, 176–188. [\[CrossRef\]](#)
- Bellini, P.; Nesi, P.; Pantaleo, G. IoT-enabled smart cities: A review of concepts, frameworks and key technologies. *Appl. Sci.* **2022**, *12*, 1607. [\[CrossRef\]](#)
- Cardoso, J.; Pereira, C.; Aguiar, A.; Morla, R. Benchmarking IoT middleware platforms. In Proceedings of the 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China, 12–15 June 2017; pp. 1–7.
- Paul, A. Graph based M2M optimization in an IoT environment. In Proceedings of the 2013 Research in Adaptive and Convergent Systems; ACM: New York, NY, USA, 2013; pp. 45–46.
- Paul, A.; Daniel, A.; Ahmad, A.; Rho, S. Cooperative cognitive intelligence for Internet of vehicles. *IEEE Syst. J.* **2017**, *11*, 1249–1258. [\[CrossRef\]](#)
- Paul, A.; Ahmad, A.; Rathore, M.M.; Jabbar, S. Smartbuddy: Defining human behaviors using big data analytics in social Internet of things. *IEEE Wirel. Commun.* **2016**, *23*, 68–74. [\[CrossRef\]](#)
- Wang, W.; Lee, K.; Murray, D. Building a generic architecture for the Internet of Things. In Proceedings of the 2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, VIC, Australia, 2–5 April 2013; pp. 333–338.
- Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Ca4iot: Context awareness for Internet of things. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications, Besancon, France, 20–23 November 2012; pp. 775–782.
- Zahariadis, T.; Papadakis, A.; Alvarez, F.; Gonzalez, J.; Lopez, F.; Facca, F.; Al-Hazmi, Y. FIWARE lab: Managing resources and services in a cloud federation supporting future internet applications. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, London, UK, 8–11 December 2014; pp. 792–799.

18. Datta, S.K.; Bonnet, C. Smart m2m gateway based architecture for m2m device and endpoint management. In Proceedings of the 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), Taipei, Taiwan, 1–3 September 2014; pp. 61–68.
19. Urbanization. J.I. W., The Free Encyclopedia. 2018. Available online: <https://en.wikipedia.org/w/index.php?title=Urbanization&oldid=822829130> (accessed on 30 January 2018).
20. Smart City. J.I. W., The Free Encyclopedia. 2018. Available online: https://en.wikipedia.org/w/index.php?title=Smart_city&oldid=823259830 (accessed on 1 February 2018).
21. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I. Towards fault tolerant fog computing for IoT-based smart city applications. In Proceedings of the 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 0752–0757.
22. Agarwal, P.; Alam, M. Investigating IoT middleware platforms for smart application development. In *Smart Cities—Opportunities and Challenges*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 231–244.
23. Pereira, J.; Batista, T.; Cavalcante, E.; Souza, A.; Lopes, F.; Cacho, N. A platform for integrating heterogeneous data and developing smart city applications. *Future Gener. Comput. Syst.* **2022**, *128*, 552–566. [[CrossRef](#)]
24. Oliveira, E.A.; Kirley, M.; Fonseca, J.C.; Gama, K. Device nimbus: An intelligent middleware for smarter services for health and fitness. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 454626. [[CrossRef](#)]
25. Jeon, S.; Jung, I. MinT: Middleware for Cooperative Interaction of Things. *Sensors* **2017**, *17*, 1452. [[CrossRef](#)]
26. Qiu, Z.; Guo, Z.; Guo, S.; Liu, Y.; Wang, Y. DAQ-Middleware: Data Acquisition Middleware based on Internet of Things. In Proceedings of the 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), Chengdu, China, 10–11 August 2017; pp. 404–412.
27. Apolinarski, W.; Iqbal, U.; Parreira, J.X. The GAMBAS middleware and SDK for smart city applications. In Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), Budapest, Hungary, 24–28 March 2014; pp. 117–122.
28. Mukherjee, B.; Neupane, R.L.; Calyam, P. End-to-End IoT Security Middleware for Cloud-Fog Communication. In Proceedings of the 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA, 26–28 June 2017; pp. 151–156.
29. Jeon, S.; Jung, I. Experimental evaluation of improved IoT middleware for flexible performance and efficient connectivity. *Ad Hoc Networks* **2018**, *70*, 61–72. [[CrossRef](#)]
30. Abreu, D.P.; Velasquez, K.; Curado, M.; Monteiro, E. A resilient Internet of Things architecture for smart cities. *Ann. Telecommun.* **2017**, *72*, 19–30. [[CrossRef](#)]
31. Zgheib, R.; Conchon, E.; Bastide, R. Engineering IoT healthcare applications: Towards a semantic data driven sustainable architecture. In *eHealth 360°. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*; Springer: Cham, Switzerland, 2017; Volume 181, pp. 407–418.
32. Rodríguez-Molina, J.; Martínez, J.-F.; Castillejo, P.; De Diego, R. SMARc: A proposal for a smart, semantic middleware architecture focused on smart city energy management. *Int. J. Distrib. Sens. Netw.* **2013**, *9*, 560418. [[CrossRef](#)]
33. Villanueva, F.J.; Santofimia, M.J.; Villa, D.; Barba, J.; Lopez, J.C. Civitas: The smart city middleware, from sensors to big data. In Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Taichung, Taiwan, 3–5 July 2013; pp. 445–450.
34. Cruz Huacarpuma, R.; de Sousa Junior, R.T.; de Holanda, M.T.; de Oliveira Albuquerque, R.; García Villalba, L.J.; Kim, T.-H. Distributed data service for data management in Internet of things middleware. *Sensors* **2017**, *17*, 977. [[CrossRef](#)]
35. Jung, H.S.; Yoon, C.S.; Lee, Y.W.; Park, J.W.; Yun, C.H. Processing IoT Data with Cloud Computing for Smart Cities. *Int. J. Web Appl.* **2017**, *9*, 88–95.
36. Cheng, B.; Solmaz, G.; Cirillo, F.; Kovacs, E.; Terasawa, K.; Kitazawa, A. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet Things J.* **2017**, *5*, 696–707. [[CrossRef](#)]
37. Chang, K.-C.; Chu, K.-C.; Wang, H.-C.; Lin, Y.-C.; Pan, J.-S. Agent-based middleware framework using distributed CPS for improving resource utilization in smart city. *Future Gener. Comput. Syst.* **2020**, *108*, 445–453. [[CrossRef](#)]
38. Akpolat, C.; Sahinel, D.; Sivrikaya, F.; Lehmann, G.; Albayrak, S. CHARIOT: An IoT Middleware for the Integration of Heterogeneous Entities in a Smart Urban Factory. In Proceedings of the Conference on Computer Science and Information System, Czech Republic, Prague, 3–6 September 2017; FedCSIS (Position Papers). pp. 135–142.
39. Bonte, P.; Ongenaes, F.; De Backere, F.; Schaballie, J.; Arndt, D.; Verstichel, S.; Mannens, E.; Van de Walle, R.; De Turck, F. The MASSIF platform: A modular and semantic platform for the development of flexible IoT services. *Knowl. Inf. Syst.* **2017**, *51*, 89–126. [[CrossRef](#)]
40. Aguilar, J.; Jerez, M.; Mendonça, M.; Sánchez, M. Performance analysis of the ubiquitous and emergent properties of an autonomic reflective middleware for smart cities. *Computing* **2020**, *102*, 2199–2228. [[CrossRef](#)]
41. Ji, Z.; Ganchev, I.; O'Droma, M.; Zhao, L.; Zhang, X. A cloud-based car parking middleware for IoT-based smart cities: Design and implementation. *Sensors* **2014**, *14*, 22372–22393. [[CrossRef](#)]
42. Huo, C.; Chien, T.-C.; Chou, P.H. Middleware for IoT-cloud integration across application domains. *IEEE Des. Test* **2014**, *31*, 21–31.

43. Al-Jaroodi, J.; Mohamed, N.; Jawhar, I.; Mahmoud, S. CoTWare: A Cloud of Things Middleware. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 5–8 June 2017; pp. 214–219.
44. Zhou, L.; Rodrigues, J.J. Service-oriented middleware for smart grid: Principle, infrastructure, and application. *IEEE Commun. Mag.* **2013**, *51*, 84–89. [[CrossRef](#)]
45. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I.; Lazarova-Molnar, S.; Mahmoud, S. SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services. *IEEE Access* **2017**, *5*, 17576–17588. [[CrossRef](#)]
46. Wang, F.; Hu, L.; Zhou, J.; Zhao, K. A data processing middleware based on SOA for the Internet of things. *J. Sens.* **2015**, *2015*, 827045. [[CrossRef](#)]
47. Khazaei, H.; Bannazadeh, H.; Leon-Garcia, A. SAVI-IoT: A Self-Managing Containerized IoT Platform. In Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), Prague, Czech Republic, 21–23 August 2017; pp. 227–234.
48. Noman, U.A.; Negash, B.; Rahmani, A.M.; Liljeberg, P.; Tenhunen, H. From threads to events: Adapting a lightweight middleware for Contiki OS. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 486–491.
49. Patti, E.; Acquaviva, A.; Jahn, M.; Pramudianto, F.; Tomasi, R.; Roubourdin, D.; Virgone, J.; Macii, E. Event-driven user-centric middleware for energy-efficient buildings and public spaces. *IEEE Syst. J.* **2016**, *10*, 1137–1146. [[CrossRef](#)]
50. Bellur, U.; Narendra, N.C.; Mohalik, S.K. AUSOM: Autonomic Service-Oriented Middleware for IoT-Based Systems. In Proceedings of the 2017 IEEE World Congress on Services (SERVICES), Honolulu, HI, USA, 25–30 June 2017; pp. 102–105.
51. Naseer, A.; Alkazemi, B.Y.; Aldoobi, H.I. TinyCO—A Middleware Model for Heterogeneous Nodes in Wireless Sensor Networks. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 259–267. [[CrossRef](#)]
52. Brundu, F.G.; Patti, E.; Osello, A.; Del Giudice, M.; Rapetti, N.; Krylovskiy, A.; Jahn, M.; Verda, V.; Guelpa, E.; Rietto, L. Iot software infrastructure for energy management and simulation in smart cities. *IEEE Trans. Ind. Inform.* **2017**, *13*, 832–840. [[CrossRef](#)]
53. Del Esposte, A. d. M.; Kon, F.; Costa, F.M.; Lago, N. InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities. In Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems, Porto, Portugal, 22–24 April 2017; pp. 35–46.
54. Kaur, N.; Sood, S.K. An energy-efficient architecture for the Internet of Things (IoT). *IEEE Syst. J.* **2017**, *11*, 796–805. [[CrossRef](#)]
55. Andročec, D.; Tomaš, B.; Kišasondi, T. Interoperability and lightweight security for simple IoT devices. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017; pp. 1285–1291.
56. Palade, A.; Cabrera, C.; White, G.; Razzaque, M.; Clarke, S. Middleware for Internet of Things: A quantitative evaluation in small scale. In Proceedings of the 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China, 12–15 June 2017; pp. 1–6.
57. Brundu, F.G.; Patti, E.; Del Giudice, M.; Osello, A.; Ramassotto, M.; Massara, F.; Marchi, F.; Musetti, A.; Macii, E.; Acquaviva, A. A scalable middleware-based infrastructure for energy management and visualization in city districts. *EAI Endorsed Trans. Cloud Syst.* **2017**, *17*, e1. [[CrossRef](#)]
58. Flask (A Python Microframework). Available online: <http://flask.pocoo.org/> (accessed on 15 September 2022).
59. Apache Spark™—Unified Analytics Engine for Big Data. Available online: <https://spark.apache.org/> (accessed on 1 August 2022).
60. MLlib | Apache Spark. Available online: <https://spark.apache.org/mllib/> (accessed on 1 April 2022).
61. MongoDB for GIANT Ideas. Available online: <https://www.mongodb.com> (accessed on 1 October 2021).
62. JSON. Available online: <https://json.org/> (accessed on 1 October 2021).
63. Android Studio and SDK Tools | Android Developers. Available online: <https://developer.android.com/studio/> (accessed on 1 October 2021).
64. Ngrok—Secure Introspectable Tunnels to Localhost. Available online: <https://ngrok.com/> (accessed on 30 September 2021).
65. Locust—A modern load testing framework. Available online: <https://locust.io/> (accessed on 30 September 2021).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.