12-18-2023 11:00 AM

Enhancing Urban Life: A Policy-Based Autonomic Smart City Management System for Efficient, Sustainable, and Self-Adaptive Urban Environments

Elham Okhovat, *Western University*

Supervisor: Michael Bauer, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Computer Science

Abstract

This thesis proposes the concept of the Policy-based Autonomic Smart City Management System, an innovative framework designed to comprehensively manage diverse aspects of urban environments, ranging from environmental conditions such as temperature and air quality to the infrastructure which comprises multiple layers of infrastructure, from sensors and devices to advanced IoT platforms and applications. Efficient management requires continuous monitoring of devices and infrastructure, data analysis, and real-time resource assessment to ensure seamless city operations and improve residents' quality of life. Automating data monitoring is essential due to the vast array of hardware and data exchanges, and round-the-clock monitoring is critical. Efficient resource use is key to cost reduction, making resource-sensitive infrastructure management crucial. This system is implemented based on the MAPE-K approach that collects the data, monitors it, analyzes it, and makes real-time decisions based on predefined policies without the need for human intervention.

The thesis introduces a novel model for an autonomic management system for smart cities, a general, end-to-end model of a smart city and delves into the algorithms and policies that underpin this system, illustrating how they interpret the data to optimize urban operations. Unique to the models is the assumption that smart cities will leverage existing platforms for IoT Management and monitoring. The autonomic management system assumes the presence of such components and leverages their capabilities. A prototype autonomic management system based on this is presented and used to demonstrate the approach. The primary objective of the Autonomic Smart City Management System is to enhance urban efficiency, sustainability, and overall quality of life for city residents, all while reducing the necessity for labor-intensive manual monitoring and management. By harnessing technology to streamline operations, this system aims to not only improve urban functionality but also result in long-term cost savings.

Keywords

Autonomic Smart City Management System, Internet of Things (IoT), Monitoring, Smart Cities, Policy-based Management

Summary for Lay Audience

An autonomic smart city management system is a computerized system that helps manage various aspects of a city from the environment, such as temperature and air quality, to the infrastructure such as the performance of computers and networks. Think of it like a "brain" for a city that uses sensors, cameras, and other technologies to collect data about what is happening in the city and manages those devices 24/7 without any human intervention. This system uses algorithms and policies to analyze data and make decisions to improve the city's operations. For example, if there is heavy traffic on a particular road, the system can automatically adjust traffic signals to reduce congestion. Similarly, if the response time of an application is higher than a threshold, the system can decrease it automatically by assigning more resources to a particular process. The purpose of an autonomic smart city management system is to enhance cities' efficiency, sustainability, and livability for their inhabitants while reducing the necessity for manual monitoring and management. By using technology to streamline operations, cities can be efficient and save money in the long run.

Acknowledgments

I would like to express my sincere gratitude to Professor Michael A. Bauer, my supervisor, for the invaluable guidance, support, and constant encouragement provided during the entire research journey. Professor Bauer's expertise, remarkable patience, and constructive feedback have significantly contributed to the development of this thesis and the enhancement of my academic capabilities. Furthermore, my deepest gratitude goes out to the esteemed faculty and staff members of the Department of Computer Science for their support.

I am also profoundly thankful to my parents and sister for their enduring support, unwavering understanding, and unceasing motivation throughout this challenging endeavor. Their love and encouragement have served as a wellspring of strength and inspiration. My appreciation also goes out to my friends, who believed in me even during the moments of self-doubt.

Contents

Abstract ii

Summary for Lay Audience iii

Acknowledgments iv

List of Tables ix

List of Figures x

Chapter 1 Introduction 1

Chapter 2 Background and Literature Review 7

2.1 Background 7

2.1.1 What is a Smart City? 7

2.1.2 IoT Platforms and Management 9

2.1.2.1 IoT Platforms 10

2.1.2.2 Management of IoT networks 14

2.1.2.3 Monitoring and Management of IoT in Smart Cities 16

2.1.2.4 Tools for Instrumenting and Monitoring 17

2.1.3 Emergence of Smart Cities 20

2.1.3.1 IoT in Smart Cities 20

2.1.3.2 Platforms for Smart Cities 25

2.1.4 Autonomic Computing 27

2.2 Literature Review 31

2.2.1 Autonomous Management 31

2.2.2 Discussion of Related Work 49

Chapter 3 Smart City Model 56

3.1 Smart City Features 58

3.2 Smart City Benefits 59

3.3 Smart City Challenges 59

3.4 A Smart City Model 61
3.4.1 A Basic Smart City Model 61
3.4.2 An Extended Model of a Smart City 64
Chapter 4 An Autonomic Management Model for Smart Cities 68
Chapter 5 Prototypes and Management Policies 77
5.1 Smart City Prototype 77
5.1.1 Smart City Prototype 77
5.1.2 Experiments with Smart City Prototype 83
5.1.2.1 Dashboard to Show the State of One Attribute for all Sensors 83
5.1.2.2 Visualization of the Telemetry Data by Sensor Location 84
5.1.2.3 Triggering Alarms and Generating Notifications. 85
5.1.3 Discussion on the Smart City Prototype 86
5.2 Autonomic Management System Prototype 89
5.2.1 Monitoring and Instrumentation Framework 92
5.2.2 Device and Infrastructure Monitoring Web Platform 97
5.2.3 Management Policies 99
5.2.3.1 Sample Policy 100
5.2.3.2 Sensor Firmware Version 101
5.2.3.3 Offline Sensor 102
5.2.3.4 Response Time 103
5.2.3.5 CPU Usage 104
5.2.3.6 Memory Usage 104
5.2.3.7 Alarms 105
5.2.3.8 Prioritization (Mission Critical vs Normal Data) 109
5.2.3.9 Power Supply 110
5.2.3.10 Calculation Based Policies 111

Chapter 6 Experiments and Evaluation 115
6.1 Experimental Configuration 115
6.1.1 Smart City Configuration 115
6.1.2 ASCMS Configuration 119
6.2 Scenario 1: Management of Sensor Attributes 119
6.2.1 Relevant Policies 121
6.2.1.1 Sensor Version 121
6.2.1.2 Sensor Priority 122
6.2.1.3 Sensor Offline 122
6.2.1.4 Sensor Power Supply 124
6.2.2 Managing Sensor Attributes 125
6.3 Scenario 2: Management of Smart City Infrastructure Using Performance Metrics 130
6.3.1 Relevant Policies 130
6.3.1.1 CPU Usage 130
6.3.1.2 Memory Usage 130
6.3.1.3 Response Time 131
6.3.2 Managing the Smart City Infrastructure 131
6.4 Scenario 3: Management of Smart City Environment and Communicating with IoT Platform. 134
6.4.1 Relevant Policies 135
6.4.1.1 Temperature 135
6.4.1.2 Wave Height 135
6.4.1.3 Parking Space 136
6.4.1.4 Water Quality 137
6.4.2 Managing the Smart City Environment. 137
6.5 Scenario 4: Sensor Diversity and Policy Variability 143

6.6 Discoveries and Reflections: Insights from Experimental Implementation 144
Chapter 7 Summary, Conclusion and Future Work 146
7.1 Contributions of the Work 147
7.2 Limitations of the Work 149
7.3 Future Directions 150
Bibliography 152
Curriculum Vitae 164

List of Tables

Table 1: Defined Policies 112

List of Figures

Figure 2.1 MAPE-K Control Loop [65] 28

Figure 2.2 Cross-Platform Channels [72] 35

Figure 2.3 Abstract Model of a Cloud-native Application Execution Environment [79]. 39

Figure 3.1 Basic Smart City Model. 62

Figure 3.2 Refined Smart City Model..... 65

Figure 4.1 Autonomic Smart City Management System (ASCMS)..... 69

Figure 4.2 Autonomic Smart City Management System Model (ASCMS) 71

Figure 5.1 Smart City Prototype 80

Figure 5.2 ThingsBoard dashboard to show the battery life of the sensors. 84

Figure 5.3 ThingsBoard time-series dashboard to show sensor telemetry data. 85

Figure 5.4 ThingsBoard alarm dashboard..... 86

Figure 5.5 Autonomic Smart City Management System Prototype 90

Figure 5.6 Node-Red service Performance Metrics..... 94

Figure 5.7 Node-Red Dashboard 94

Figure 5.8 ThingsBoard (SpringBoot) Service Performance Metrics 95

Figure 5.9 ThingsBoard (Spring boot) Dashboard 96

Figure 5.10 ThingsBoard (Database) Service Performance Metrics 96

Figure 5.11 ThingsBoard (Database) Dashboard..... 97

Figure 5.12 Device and Infrastructure Monitoring Web Platform 99

Figure 5.13 Policy Form 99

Figure 6.1 Experimental Configuration 116

Figure 6.2 Sensor Attributes for “Sensor 39” before action. 126

Figure 6.3 Managing the Battery Level. 127

Figure 6.4 Managing the sensor when it is offline and has high priority. 128

Figure 6.5 Managing sensor version. 128

Figure 6.6 Sensor attributes after the ASCMS changes. 129

Figure 6.7 Managing CPU Usage. 132

Figure 6.8 Managing Response Time while the ASCMS is working. 133

Figure 6.9 Managing Memory Usage while the ASCMS is working. 133

Figure 6.10 The sequence of policy violations over time. 134

Figure 6.11 Temperature measurement for “Temperature Sensor 1”. 137

Figure 6.12 Managing Temperature while the ASCMS is working. 138

Figure 6.13 Temperature after action execution. 139

Figure 6.14 Wave height information before the action. 139

Figure 6.15 Managing wave height while ASCMS is working. 140

Figure 6.16 Wave height after action execution 140

Figure 6.17 Parking information before the action. 141

Figure 6.18 Parking sign when the parking lot is not full. 141

Figure 6.19 Parking information before the action. 141

Figure 6.20 Parking sign when the parking lot is full. 141

Figure 6.21 Turbidity information when the level is below the threshold. 142

Figure 6.22 Turbidity information when the level is above the threshold. 142

Figure 6.23 Alert when turbidity is above the threshold. 142

Figure 6.24 The sequence of policy violation over time for smart city environment management. 143

Figure 6.25 The sequence of policy violation over time for scenario 4. 144

Chapter 1 Introduction

Recent advances in electronics and wireless communications have enabled the design and construction of sensors with low power consumption, small size, reasonable price, and created a variety of uses [1]. This rapid development of technology has also inspired the idea of the "smart city" and made "smart city" initiatives the mainstream of much urban activity. The advances in sensors and related technology are broadly referred to as the Internet of Things (IoT). IoT refers to a network of devices such as sensors, wearable gadgets, and anything that has the ability of connecting to other devices, that send and receive the data and communicate with other devices through communication protocols.

There are various definitions for IoT, and they can provide somewhat different points of view. Zanella et al. [1], defined IoT as an architecture of a massive number of heterogeneous devices and end systems that can provide open access to datasets for service development. The authors note that because of the diversity of devices, technologies, and services that may exist in a smart city, creating this architecture is extremely complicated. They also consider the technologies and protocols in an urban IoT to support the smart city and its administration. They implemented the Padova smart city project to collect environmental data such as temperature, humidity, carbon monoxide level, etc. and also monitor the streetlights operation. Gubbi et al. [2] defined IoT as an environment of Wireless Sensor Network (WSN) technologies that allow the measurement of environmental factors. Their IoT network consists of WSN devices such as embedded sensors and actuators that exist in the environment and share the information to create a Common Operating Picture (COP). Our definition of the Internet of Things (IoT) in this research is a combination of interconnected objects with a unique identifier such as sensors, vehicles, smartphones, home appliances, wearable gadgets and even medical equipment that can interact with each other, send, or receive data through the internet or other communication technologies like MQTT, COAP, etc. This definition includes a large number of heterogeneous connected nodes.

A smart city is a city that benefits from the Internet of Things capabilities to improve the life of its citizens. In a smart city, IoT is used to provide connectivity among all the entities and

devices within the city. The entities can be software and hardware elements and a smart city may contain tens of thousands of these elements. According to a prediction published by the International Data Corporation (IDC) in 2020, the number of connected devices around the globe would be 55.7 billion by 2025 and 75% of which will be used in an IoT ecosystem [3]. This number of devices produce a huge amount of data and monitoring that data in real-time is a demanding task. Aside from that, any smart city network and IoT infrastructure need to be monitored to ensure that everything works smoothly and efficiently to achieve their goal - which is providing services and resources to their users.

Working with smart cities introduces distinctive challenges and considerations compared to conventional systems. Unlike isolated systems, smart cities involve a complex, interconnected web of diverse elements, including infrastructure, technology, and community dynamics. The sheer scale and diversity of data sources within a smart city present unique challenges in terms of data management, and analysis. Additionally, smart cities often require interdisciplinary collaboration, as the integration of various technologies and services necessitates expertise in urban planning, IoT, data science, and more. Furthermore, the dynamic nature of urban environments, with continuous changes and unexpected events, demands adaptable and resilient systems. In navigating these complexities, it becomes increasingly evident that an autonomic, policy-based smart city management system is not just beneficial but essential. Such an advanced system provides a framework for automating decision-making processes, responding dynamically to changing conditions, and ensuring efficient management of urban resources. It is crucial to emphasize that our work is built upon existing foundations, particularly leveraging advancements in monitoring and IoT platforms.

In this research, we assume that the smart city infrastructure has several layers: from sensors and devices at the base tier to complex IoT platforms, third-party applications, and managerial units at the upper tiers. Large-scale smart city management is a challenging task because the IoT environment is subject to a lot of uncertainties. Unpredictable natural disasters, wireless communication problems, radio interference, failure of the IoT nodes, resource limitation, calibration of sensors, dynamic network topology, and even excessive network traffic during a data overload can be some of the uncertainties in the IoT environment that should be

considered during smart city management. Management requires data collection and analysis, device tracking, determining resource status in real-time, deploying new versions of software, determining performance problems or device failures, reconfiguration of connections and applications on failures or for performance reasons. Management needs to produce the proper action plan in accordance with resources' status and available actions to ensure that the smart city infrastructure can continue to operate and support the services and overall quality of life for the city's residents.

Due to the massive number of heterogeneous hardware and software elements, and protocols in a smart city and the huge amount of data transferring between the devices, and the shortage of financial, technical and personnel resources, monitoring, analysis, and management of data should be done as automatically as possible to limit the human resources needed to maintain the infrastructure. Since problems in a smart city might happen during the night or on holidays, it is critical that monitoring and management be done constantly 24/7 to avoid problems. Further, the optimal use of resources in a smart city is very important in order to reduce capital and operating costs, so management of the infrastructure should also be sensitive to how resources are being consumed.

Our research focuses on autonomic management methods to support the management of smart city infrastructure and environment. Such an approach requires the collection of operational data about the elements of a smart city infrastructure. This data is operational data and would be in addition to the data that the components, e.g., the IoT devices, in the smart city, would be generating. We will focus on the creation of policy-based autonomic methods that use this operational data to manage aspects of a smart city. Using policy-based methods for this purpose makes autonomic management more straightforward by focusing on defining the policies and adjusting them.

The creation of a policy-based autonomic management system for a smart city requires that we address a number of important questions:

- How can we monitor the performance of the operational side of the smart city infrastructure to ensure it works well as a whole e.g., the connections, response time, etc.?

- How can we decrease human intervention in smart city management?

- How can we efficiently and automatically monitor and manage the resources within the smart city infrastructure, e.g., sensors, bandwidth, storage, CPU, memory, etc.?

- What is an appropriate architecture for an autonomic system that is an integral part of a smart city infrastructure?

- What are the autonomic services required for managing aspects of a smart city?

- Assuming a policy-based approach, what kinds of policies are needed to manage the smart city infrastructure?

Our research addresses these questions. The contributions of our work are:

1. **A Comprehensive Model of a Smart City Infrastructure:** Previous work has considered different aspects of a smart city infrastructure, but none have considered addressing the management of the end-to-end infrastructure, i.e., from sensors to hosts and applications. We introduce a model that captures the entire infrastructure of a smart city and considers how to manage many aspects.
2. **Models Which Leverage the Presence of Existing Components:** There has been significant work on platforms, applications and tools to support IoT and analysis of sensor data. Current work on smart cities makes use of such tools and so it is likely that such tools will be used in the future. We assume that this is the case and incorporate central ones into our model of a smart city and into our model of autonomic management. In particular, we assume:
    a. An IoT platform that is used for managing of and interaction with sensors within the smart city.
    b. The presence of a data filtering component as part of the data collection process from sensors.
    c. The presence of a performance monitoring component as part of the autonomic management system.

These types of components are becoming commonplace, especially IoT platforms and applications for filtering data. It is reasonable to expect that smart cities will leverage these and that autonomic management methods must be able to leverage these as well. We have introduced a model for autonomic management with this in mind.

3. **Real-time Sensor Data Insights:** Our work delves into the real-time monitoring of sensor attributes and measurements that are complementary to what an IoT platform may provide demonstrating the integration of the IoT platform with our model of autonomic management.
4. **Policy-Based Autonomic Management Model:** We present a policy-based model for the autonomic management of smart cities.
5. **Management Interface:** We introduce a management interface providing administrators with invaluable insights into action execution. This empowers them with the necessary information to ensure that the smart city operations are in alignment with the high-level goals.
6. **Demonstration of the Utility and Scope of the Models**: Based on our models, prototypes for smart city infrastructure and the autonomic management system are presented and used to demonstrate the effectiveness of the models and advantages of autonomic management.

These contributions collectively advance the field of smart city management, paving the way for more efficient, autonomous, and data-driven urban environments.

The remaining sections of this thesis are structured as follows. In Chapter 2, we will delve into the background of IoT management, monitoring and management of smart city environments, autonomic computing, and autonomous management. We will also review relevant literature in these areas to provide a comprehensive understanding of the research domain.

Chapter 3 introduces our model of a smart city and the various components that we have integrated to create an enhanced smart city model. This chapter will lay the foundation for the rest of the thesis and provide readers with a clear understanding of the theoretical framework we have developed.

Moving on, Chapter 4 will present our model of an autonomic smart city management system (ASCMS) and provide a detailed explanation of its internal architecture. This Chapter will be instrumental in helping readers grasp the core concept of our research - an autonomous system that can manage a smart city environment and infrastructure.

Chapter 5 will showcase the prototypes of the smart city and the autonomic management system. In particular, we introduce our substantive autonomic management system prototype which captures the key aspects of our proposed ASCMS. We will also provide a rationale for our technology choices and explain why each component was selected for the prototypes and describe the operational aspects of our prototypes. In this Chapter, we will also present the policies that we have defined in the ASCMS. We will provide a detailed explanation of each policy to ensure a comprehensive understanding.

In Chapter 6, we will showcase a range of examples and experiments conducted using a prototype autonomic smart city management system (ASCMS) and assess its performance. We will illustrate the state of the smart city before and after the ASCMS makes changes based on policies, highlighting how it effectively manages the entire smart city ecosystem.

To conclude the thesis, Chapter 7 will serve as a comprehensive summary of our findings and contributions. Subsequently, we will elucidate the limitations inherent in our research. It is crucial to acknowledge these constraints as they offer valuable insights into the scope and potential directions for future endeavors. By openly addressing the limitations of our work, we aim to contribute to the ongoing dialogue surrounding the development and refinement of smart city management systems. Finally, we will explore potential avenues for future research in this field, fostering opportunities for further exploration and advancement.

Overall, this thesis aims to introduce a comprehensive system for autonomic smart city management and demonstrate it through a substantive prototype in order to show its potential to enhance the sustainability, efficiency, and livability of smart cities without human intervention.

Chapter 2 Background and Literature Review

In this Chapter, we present the key concepts and research background. Subsequently, we undertake a thorough exploration of existing literature pertaining to autonomous management. We conclude the Chapter by discussing what challenges are effectively addressed in the domain of autonomous management. Concurrently, we deliberate on the challenges that still need resolution. This reflective analysis not only underscores the advancements achieved in mitigating specific issues but also illuminates the dynamic landscape, pinpointing areas where further research and innovative solutions are imperative for continued progress.

2.1 Background

This section starts with a review of smart city definitions and describes what we mean by this term in our research. We then review previous research on smart city data management. Because the Internet of things is the centerpiece of any smart city, we then review the definition of IoT and its management to get a sense of what IoT means and what requirements should be satisfied to manage an IoT network. Also, we will clarify how each of the requirements relates to each other. After reviewing the requirements for the management of IoT, we examine several IoT platforms, their characteristics, and features. After that, some of the tools and platforms for software instrumentation and monitoring will be explored. This section ends with an examination of research on autonomic computing.

2.1.1 What is a Smart City?

There is not a rigid definition for a smart city because it depends on how one looks at this concept. The following provides some different definitions of what different researchers perceive as smart cities that are relevant to our research.

The term “Smart City” was coined by Cisco and IBM [4] to describe a city that benefits from information and communications technology and automation. This term was used to refer to “a city that makes a conscious effort to innovatively employ information and communication technologies (ICT) to support a more inclusive, diverse and sustainable urban environment”.

Bakıcı et al. [5] defined the smart city as a high-tech and advanced physical environment that provides a connection between citizens, information and city elements to increase the quality of life for its citizens, create a more sustainable, efficient and transparent public administration, and promote innovation and facilitate access to the information. This paper mainly focuses on Barcelona's smart city model and explores the main components of the Smart City strategy of Barcelona.

According to the ISO report published in 2015 [6], a city is a system of systems that has a specific history, environmental and societal context. But when it comes to a smart city it should reach its goals by using the resources in an efficient and consistent manner. Not only should all of the components of the smart city such as people, infrastructure, institutions, finances, facilities etc. work effectively but also, they should also cooperate in a harmonious and smooth way to help the city flourish and promote growth and innovation in the city.

In Zygiaris's work [7], the term "smart city" has a general meaning of an IT-based urban ecosystem and includes concepts such as greenness, openness, intelligence, and innovation, with the goal of environmental and social sustainability.

Another definition for the smart city which is provided by Schleicher et al. [8], suggests that smart cities are cities that use communication technologies to provide their people with services and use information technology to make the use of their resources smarter and more effective.

Fernandez-Anez [9] surveys the definitions of a smart city from different stakeholders' perspectives and at the end provides a holistic definition for it:

"A Smart City is a system that enhances human and social capital wisely using and interacting with natural and economic resources via technology-based solutions and innovation to address public issues and efficiently achieve sustainable development and high quality of life on the basis of a multi-stakeholder, municipally based partnership." [9]

Smart cities, as defined by these papers share commonalities in their emphasis on technology-driven urban development, efficiency, sustainability, and the enhancement of citizens' quality

of life. Technology, particularly information and communication technologies (ICT), is consistently highlighted as a key enabler. Efficiency and sustainability are overarching goals, with an emphasis on the interconnectedness of various urban elements. Smart city definitions vary in their focus areas, with Bakıcı et al. [5] exploring Barcelona's model and Zygiaris [7] emphasizing greenness, openness, intelligence, and innovation. This diversity underscores the multidimensional nature of the smart city concept. The ISO report [6] contributes a broader perspective, framing a city as a system of systems with historical, environmental, and societal dimensions, highlighting the complexity and interconnectedness of urban systems. Fernandez-Anez [9] takes a holistic approach, incorporating perspectives from various stakeholders to underscore the importance of human and social capital, multi-stakeholder partnerships, and municipal involvement in shaping the understanding of smart cities.

What we consider a smart city is a city that benefits from IoT to provide connectivity between its elements and manages them in an efficient and optimized way. Thus, we assume the presence of and use of an IoT infrastructure, a network and computing infrastructure for data collection and applications important to the smart city, and a management system that monitors and manages the IoT and computing infrastructure of the smart city. Our main focus is on monitoring and management of a smart city from the environment to the infrastructure.

Our review of related work first starts by considering IoT platforms and then how such platforms are managed. We then consider smart city platforms followed by approaches for management of smart city infrastructure. We conclude by examining work on Autonomic Computing and its use in smart cities.

2.1.2 IoT Platforms and Management

Our view of a smart city is one that makes extensive use of IoT. In this section, we review previous work on the development of IoT platforms. These platforms are software environments that support the use of sensors and devices and a number have been developed with the use in smart cities in mind. We also look at work related to the management of IoT environments in general and look at more specific work focused on IoT management in smart

cities.

2.1.2.1 IoT Platforms

Any IoT network consists of IoT devices that are connected to other devices and applications and transfer the data over the internet. An IoT platform can be looked at as a middleware between the devices and end-users which can connect the devices and sensors, collect, store their data, manage devices and users, and provide data visualization. Some of the more advanced platforms can even trigger alarms based on the data, control the actuators and devices in the network and support white labelling and multi-tenancy. There are many tools, services, and platforms for IoT and smart city management. Several solutions have been investigated as described briefly in this section.

Snap4city [10] is an Open-source IoT/IoE platform that is proposed by DISIT Lab in response to providing the functional and non-functional requirements for the IoT network. Functional requirements are providing a platform for several operators, supporting varied real time communication, allowing users and stakeholders to create their desired applications, and managing the data. On the other hand, non-functional requirements are scalability, standard compliance, robustness, distributed, heterogeneity, interoperability, security, and privacy. This platform benefits from microservice architecture and includes a variety of components that are responsible for data collection, data storing and management, creating IoT applications and services, executing and controlling, representing the data, and providing access to data and services. This solution can support heterogeneous data and provides data analytics and insight into the city's condition through dashboards [11].

FIWARE [12] is another cloud base framework that supports the development of smart solutions such as smart cities, smart industry, smart health, and so on in a cheaper and faster way. It provides open-source software platforms that can be joined to other third-party components to create complex applications. This platform can capture and process big data and convert it into knowledge. FIWARE Lab has five components: i) Context processing, analysis and visualization, ii) Core context management, iii) Interface to IoT, robotics and

third-party systems, iv) Data/API management, publication, and monetization, and v) Deployment tools.

Orchestra cities [13] is an open-source, multi-function smart city platform that provides an environment to connect devices and citizens and collect, analyze, and share data. This platform is made from microservices and can support multiple communication protocols and data formats. Orchestra cities is an extension of the FIWARE platform and other than citizens and individuals, can be used by industry and the public sector.

Fiwoo is a European data driven open IoT platform based on FIWARE [14] that allows users to create and design an IoT environment. Because of its simple user interface, the user can add their devices, create plans, and manage the devices, data and applications in real-time without any computer knowledge. This platform supports smart cities, smart ports, smart buildings, and smart industry [15].

Another IoT platform that supports smart cities in real-time is thethings.io [16]. This platform is a serverless solution and it can support most of the existing protocols from HTTP to MQTT and CoAP. thething.io provides cloud code processing, action management, data monitoring and visualization, and uses AI to analyze the data and give insights and improve the users' products. It also offers some advice for monitoring and tracking the environment and machines.

OpenIoT [17]is an open service framework written in Java that supports cloud-based services and uses a SaaS delivery model. This framework can be used for smart cities as well. The OpenIoT structure makes it easy not only for developers to create their services to keep up with the fast-paced growth of the Internet of Things ecosystem, but it also helps users to search for what services they need and use them in their products and applications [18]. OpenIoT can be used for device provisioning and management. This framework does not provide event detection and data analytics which are beneficial in the IoT environment [19]. In a paper written by da Cruz et al. that assessed some of the most popular IoT platforms it is said that OpenIoT does not comply with the security requirements [20].

The next open-source IoT platform is Thinger.io [21] which is hardware agnostic and provides a console that can process, monitor, and manage a huge amount of data in a cloud environment. Thinger.io supports modelling and implementing data fusion applications and allows remote monitoring through dashboards. Its usability ranges from smart buildings to smart grids and infrastructure. The free version of this platform can support up to 2 devices and provides some basic features such as dashboards, endpoints, and device management capabilities, but the platform does not enable management of the assets (such as buildings, regions, etc.), projects, accounts, etc. [22].

Mainflux [23,24] is an open IoT cloud platform written in GO that supports secured connections over various protocols such as TLS, DTLS, MQTT, WebSocket, and so on. Like other IoT platforms, Mainflux can also offer device management and provisioning, access control, event management, and analytics. One of the main advantages of this platform is that it can supply logging and instrumentation through OpenTracing and Prometheus. Its architecture is based on microservices containerized by Docker and deployed by Kubernetes.

IoTivity [25] is a Korean-based open source IoT framework that is implemented based on Open Connectivity Foundation (OCF) standards and is written in C and Java. This framework works cross-platform because it has abstract interfaces that can be used to interact with the operating system. IoTivity has the ability to find nearby devices, send and receive the data, and manage the devices and the data in an IoT ecosystem. One of its drawbacks is that it only supports CoAP, not the other protocols [26,27].

LinkSmart [28] is a free European semantic IoT middleware platform inspired by the Hydra project [29] that has a modular architecture and allows the fast development of smart applications for connecting heterogeneous devices. This platform extends the Hydra project by combining semantic web service and SOA principles to provide syntactic interoperability to the application level. Its architecture consists of device integration and abstraction layer, service provisioning, data management and processing, network and security, and human-computer interaction. This platform supports smart city, smart building, smart energy, and industry 4.0. Its architecture consists of device integration and abstraction layer, service provisioning, data management and processing, network and security, and human-computer

interaction [30]. Da Cruz et al. claimed that LinkSmart does not offer device authentication, MAC, and IP storage per device, does not support communication methods other than the REST and its response time is high which is not good in a network with so many devices and huge amount of data [20].

ThingsJS [31, 32] is a distributed IoT middleware written in JavaScript that uses a publish/subscribe (server/client model) communication paradigm and is able to run, manage and schedule JavaScript programs on heterogeneous devices. ThingsJS scheduler design benefits from Machine Learning to predict the execution time and SMT-based solver to optimize the execution time by considering the resource limitations. This framework also provides APIs and services for developers and monitors information about CPU and memory through a dashboard created by Express.js[1] and react.js. Moreover, ThingsJS helps the migration of JavaScript programs and even process execution between the physical devices in the IoT environment. This platform can change device settings based on the sensor measurements. They describe a scenario that illustrates how the system can manage the temperature using the publish/subscribe model.

ThingsBoard [33] is a flexible and scalable open-source IoT platform that is written in Java and can offer a wide variety of capabilities that are needed in an IoT network such as provisioning and management of sensors and devices, data collection and analysis, visualizing the real-time data with widgets on the dashboards, triggering alarms based on the data fluctuations, controlling the devices, and so many more. This platform is very scalable and customizable and can support different standard IoT communication protocols, like MQTT, COAP and HTTP, also it can define and manage users, customers, devices, assets, alarms, and dashboards. Because ThingsBoard deployment is also very flexible, it not only can be launched on a local computer but also can be deployed on cloud services such as AWS, Azure and so on. Also, ThingsBoard offers two types of architecture: Monolithic and Microservices. Finally, based on the amount and type of the data, it provides different options like SQL

[1] Express.js or Express is a back-end web application framework for Node.js; https://expressjs.com/.

(PostgreSQL, HSQLDB), NoSQL (Cassandra) and hybrid approach which stores all the data related to ThingsBoard itself in PostgreSQL and time-series in Cassandra or timescale DB.

There are numerous platforms available for managing and monitoring IoT networks, each with their own set of capabilities and features. However, after careful evaluation and analysis, we decided to choose ThingsBoard as the platform for our research. One of the key reasons for this decision is that ThingsBoard is an open-source platform, which means that it can be freely accessed, modified and customized by users. This makes it an extremely flexible and adaptable platform that can be tailored to suit the specific needs of different users and applications.

Additionally, ThingsBoard is highly scalable and can easily handle large volumes of data, making it ideal for use in smart city scenarios where vast amounts of data need to be collected and analyzed. The platform also offers a range of advanced features and capabilities, such as real-time data visualization, rule engine, device management, and multi-tenancy support, among others. These features make it easier for users to manage and monitor their IoT networks and devices, and to make sense of the data generated by these devices.

Overall, we believe that ThingsBoard is the ideal platform for our research, as it offers a wide range of capabilities and features that are essential for effective IoT network management and monitoring. Moreover, its open-source nature makes it a highly customizable and adaptable platform that can be tailored to meet the specific needs of different users and applications.

2.1.2.2 Management of IoT networks

Because there are massive numbers of heterogeneous devices and technologies in an IoT network, it needs to be managed in an efficient way. IoT device management includes the tasks and operations to support IoT solutions. This management has various aspects:

1. Provisioning and Authentication: the management system should enable the devices to enroll into the system after verification of their identity by checking their credentials. Because in a smart city and IoT network delays should be avoided, in [34] the resource provisioning for IoT applications for Antwerp’s City of Things is proposed and it uses Integer Linear Programming and Fog Computing paradigm to decrease latency and increase energy efficiency.

2. Configuration and Control: To increase the IoT network performance and functionality, we need to configure or control the device settings or even restart the deployed devices.
3. Monitoring and Diagnostics: Another task of a management system is to provide logs needed to monitor the network and detect bugs and faulty operations.
4. Software Maintenance and Updates: Sometimes the reason behind issues, bugs or security problems in the network can be outdated device firmware. Therefore, one of the most important tasks of the management system is to be able to update the devices and maintain the overall network, and this action should take place remotely because of the high number of devices. [35]
5. Security and Privacy: Also, the management system should ensure that the IoT network is safe and secure and there is no threat to the infrastructure.

Gürgen, et al. [36], consider the management of the sensing devices to have the goal to improve the efficiency of the entities and it includes configuration, monitoring, and administration of them. They consider management to encompass four areas: network management, system management, application management, and device management. They proposed a common management framework for networked sensing devices and formed their approach based on three crucial characteristics of the management categories which are functional areas, hierarchical architectures, and management operations on data models. For the management function, they focused on configuration management, performance management, and software management. Their solution consists of three levels of architecture which are the most common ones among all management solutions: managers, agents, and managed entities. Finally, management operations include GET, SET, NOTIFY and ACT. In this research, the focus is mostly on providing some suggestions and recommendations on networked sensing device management and they do not talk about a specific management framework.

Ersue et al. [37] provide some requirements for the management of constrained devices such as system management, protocol management, configuration management, functionality monitoring, self-management, access control and security management, energy management, traffic management, etc. The reason for reviewing this paper is that in an IoT network we deal

with so many devices that have constraints regarding CPU, memory, power and even bandwidth. Each of these categories has some sub-management tasks that are crucial to ensure that a specific management requirement is met. For example, the system management should offer features such as supporting multiple devices in a network, guaranteeing scalability, providing hierarchical management, etc.

Aboubakar et al. [38] categorized IoT network management into two groups, traditional network management, and IoT low power network management. Then, they highlighted six operations that a traditional management system should provide notably: configuration management, topology management, security management, QoS management, fault management, and network maintenance and troubleshooting. Next, they continued by listing the challenges that exist in an IoT low power network such as device heterogeneity, dynamic network topology, resource limitations, and unreliable radio links. Subsequently, the requirements of IoT low power network management are identified to deal with the aforementioned challenges. Some of these requirements are similar to the traditional management requirements but some requirements specifically target low power networks including scalability, energy efficiency, security and self-configuration.

Effective management in an IoT network is critical to ensuring the smooth functioning of a smart city. Such management involves several tasks such as device provisioning and authentication, configuration, and control of devices, monitoring and diagnostics, network recovery, and software maintenance and updates. As IoT forms the core of a smart city network and enables the monitoring and management of smart city devices and sensors, it is imperative to select a reliable and advanced platform that offers the above-mentioned features. The selection of such a platform assumes importance as devices require a platform to communicate with each other and the cloud.

2.1.2.3 Monitoring and Management of IoT in Smart Cities

When it comes to monitoring in an IoT network, there are two main approaches that can be taken. The first approach is centered around monitoring the sensors and devices that are part of the network, which entails gathering real-time data from these devices and closely

monitoring them. This type of monitoring is important because it provides valuable insights into the state of the network and allows for prompt action to be taken if any issues are detected.

The second approach to monitoring in an IoT network is focused on infrastructure monitoring, which involves keeping track of the various platforms and components that make up the smart city ecosystem. Infrastructure monitoring in an IoT network is crucial to ensure the smooth operation of the entire system and it provides key metrics that allow for the performance of the network to be analyzed and optimized. It is through infrastructure monitoring that network administrators can gain a comprehensive understanding of the network's overall health and take steps to maintain or improve it. Infrastructure monitoring can help detect any issues or anomalies in the system, such as server downtime, network congestion, or security breaches, and take appropriate action to resolve them. Moreover, monitoring the usage of resources such as CPU, memory, and disk space can help optimize the utilization of the resources and avoid system overload. Therefore, it is essential to have a robust infrastructure monitoring system in place to ensure the reliability and scalability of the IoT network.

In this section, we will explore some of the key metrics that are typically used to monitor the state of an IoT network's infrastructure. These metrics can include everything from network uptime and response time to system load and utilization rates. By tracking these metrics and analyzing the data they provide, network administrators can gain insights into the performance of the network and identify areas where improvements can be made.

2.1.2.4 Tools for Instrumenting and Monitoring

In the realm of performance monitoring for IoT networks and applications, there is a plethora of available tools and platforms. However, for the purpose of this section, we will examine a handful of the most widely recognized tools in the field. By doing so, we can gain insight into the features and capabilities offered by these tools and better understand their suitability for managing IoT networks. Ultimately, we will present our selected platform as the most suitable for our needs based on our assessment of the available options.

Zipkin [39] is an open-source distributed tracing system designed to trace the application and measure the time for each service in a path and detect any failed operation in the application

or microservices architecture. Its architecture consists of 4 different components, Collector, Storage, Query service and Web UI. It supports various programming languages and integrates with other observability tools, making it a valuable tool for monitoring and optimizing distributed systems. To utilize Zipkin and incorporate it into your application, you need to include the Zipkin library in your platform.

Unlike Zipkin which focuses on tracing, Prometheus [40] is for monitoring the system and collecting the metric data from a distributed environment. Six main components are involved in Prometheus architecture: Standalone server, Client libraries, Push gateway, Exporters, Alert manager, and Support tools. Prometheus follows a pull-based model, where it periodically retrieves metrics from configured targets such as applications, services, or even the underlying infrastructure components. This toolkit can collect and store the metrics as time-series data in its own database and it is written in Go.

OpenCensus [41] is a comprehensive instrumentation platform that allows measuring, collecting, and exporting not only the time-series metrics but also distributed traces of a target microservice or even a monolithic application. The advantage of this tool is that it is flexible and not dependent on a specific system or software. It can provide the metrics about the application architecture, capture the time for each service and show how a request navigates through the services. This tool supports many languages such as C#, C++, Java, Python, and so on and it also benefits from Prometheus to collect the metrics and Zipkin or Jaeger [42] to trace the application. Unfortunately, this tool is no longer supported by the team.

OpenTelemetry [43] is a vendor-neutral observability framework and a combination of OpenCensus and OpenTracing [44]. This platform offers two types of instrumentation: manual and auto instrumentation. This platform is like OpenCensus and it captures architecture metrics and distributed tracing data and sends them to the chosen backend such as Prometheus, Zipkin, Jaeger, Google cloud and so on. It can support a variety of programming languages and offers a library for Go, Java, Python etc.

Datadog [45] is a service that can monitor servers, applications and other components in infrastructure and provide some helpful information about that infrastructure such as

performance metrics, traces, and logs. This software can be deployed on-premises or as SaaS and it is compatible with almost all of the operating systems. It supports cloud platforms like AWS, Red Hat OpenShift, Azure and google cloud. Using this platform, you can plan what features you need, and choose the services based on the monitoring scale. For example, you can select some features such as infrastructure, log management, database monitoring, synthetic monitoring, user monitoring, network monitoring and so on [46].

New Relic [47] is also a monitoring cloud-based platform that can gather metrics, logs and traces from software and similar to Datadog has so many capabilities like application monitoring, infrastructure monitoring, browser monitoring, network monitoring, etc. This platform has so many agents to monitor different technologies and languages such as android agent, iOS agent, java agent, python agent and so many other helpful ones.

Grafana [48] is a tool that helps people analyze and visualize data from different sources. It has a user-friendly interface with customizable dashboards, so users can create interactive visualizations that make it easy to understand their data. It also can be connected to various types of data, like databases and cloud platforms. It provides a query editor allowing users to write queries and fetch data from these sources. Grafana possesses the capability to establish alerts and notifications based on predetermined conditions, allowing for heightened awareness of critical events. Grafana is widely used in the DevOps and monitoring communities because it helps monitor system performance, spot trends, and make informed decisions based on real-time data.

Dynatrace [49] is a cloud monitoring software intelligence platform that provides full environment observability and can keep track of application performance, availability, stability, and behaviour and optimize them in real time. This Application Performance Management (APM) platform can extract the dependencies inside an environment and increase productivity by providing operational insights. It can also give a very holistic real-time view of what is going on within the environment and application through metrics and provide the application traces and instrumentation information. This solution is a scalable solution that can support the application and services installed on either a local computer or on the cloud platforms from AWS and Google Cloud Platform to Microsoft Azure and

Kubernetes. Dynatrace proposed an AI agent called Davis AI which can detect the diagnosis problems automatically in the environment and notify the administrators about them. Dynatrace offers two types of deployment: SaaS and Managed. In this research, the SaaS version is used. For performance monitoring, we use this platform because of its superior ranking on the Gartner website, with an overall score of 4.5 [50]. The platform's remarkable ratings in Integration & Deployment (4.6) and Product Capabilities (4.6) further solidified its appeal. It excelled across various factors, including monitoring servers (4.7), network (4.3), storage systems (4.4), databases (4.4), hypervisors (4.3), scalability (4.6), integration (4.4), customization, and ease of deployment, administration, and maintenance (all rated at 4.4). Moreover, the endorsement from 90 percent of peers underscored its effectiveness. Having worked with other platforms, Dynatrace stood out by delivering the specific information we needed, encompassing service and process metrics, as well as host metrics. Also, it gives the flexibility we are looking for and it provides intelligent observability [51] which means we are able to not only get the traces and data flows in an application but also extract logs, metrics and other information about monitoring of the whole smart city environment.

2.1.3 Emergence of Smart Cities

There have been many different research efforts related to the idea of smart cities. In the following, we first outline work that has made the utilization of the Internet of Things (IoT) in urban environments to enable the development of smart cities. This involves leveraging IoT technologies to enhance various aspects of city life, such as transportation, energy management, public safety, and environmental sustainability. We then describe platforms developed for building smart city infrastructure where we focus on those that are most closely aligned with our work.

2.1.3.1 IoT in Smart Cities

IoT in smart cities is seen as an essential element in order to monitor a wide range of physical and environmental aspects of a smart city and in some cases enable changes to be made. In the following, we review a number of efforts on the use of IoT to monitor and support activities within cities.

Sakhardande et al. [52] developed a disaster management and smart city monitoring system that employs hardware components rather than software. Sensors, actuators, WiFi adapters, power supplies and Arduinos are all found in each module of the system. The network topology is based on the star and every node has its own unique identifier. The system has two modes: monitoring and disaster management. The system gathers data in the first mode and transmits it to certain cloud servers for processing. When a node receives the alert signal, it terminates its monitoring task and switches to disaster management mode. This node then transmits that signal to other networks that it is connected to and this process continues until the signal reaches the nodes that are located in the affected area. For a smart city with a huge number of sensors, using an Arduino in each module does not seem feasible.

Suakanto et al. [53] developed a dashboard to present the measurement information of the state of the city of Bandung, Indonesia in real-time. This measurement information contains temperature, air and water quality, traffic, and so on. Their proposed system architecture consists of the sensors deployed in the city for data collection, and processing servers that receive the data from the sensors through Remote Terminal Unit(RTU) and over the internet. The unit responsible for displaying sensor data and the present state of the city is the user application dashboard. However, the proposed solution is limited to monitoring and presentation of data; there is no consideration of managing the infrastructure.

A real-time data processing platform called My City Dashboard is introduced by Usurelu et al. [54] to process real-time data such as temperature and noise in a smart city. This platform is developed to support scalability, modularity, and pluggability. The platform comprises of four layers, each utilizing a diverse range of technologies. For example, the acquisition layer employs technologies such as Apache Kafka and RabbitMQ, while the processing layer makes use of Apache Spark and Flink. The persistence layer incorporates MongoDB and PostGIS modules for PostgreSQL. The dashboard layer, on the other hand, is constructed based on Service Oriented Architecture (SOA) and RESTful services. The dashboard that is provided by this solution represents the city map that is divided into tiles and for each tile, statistics are shown, such as the average, maximum and minimum. This solution focuses on the sensor data

and monitoring the state of each stream type, like noise, temperature and pollution, for a specific tile.

An IoT architecture called Souly is proposed by Dryjanski et al. [55] that focuses on monitoring and management of smart buildings and hotels as part of the IoT environment. This system consists of sensors and actuators, IoT gateway, a cloud platform, an administration panel, API modules and a mobile application. The main component of the system is a cloud platform which is responsible for data collection and processing, as well as providing reports and data monitoring. The data is stored in two databases, one for storing short-term data that can be used for monitoring purposes and the other database that stores room configuration and tenant information. This system uses an MQTT broker and receives the data using this protocol. Souly can provide information about the state of the devices in the building through the mobile and web panels as well as logs related to problems in memory allocation, applications and even unsuccessful logins. This architecture is also capable of triggering notifications regarding device failures.

In [56], De Paolis et al. used the ThingsBoard IoT platform[33] and Spark to collect and analyze data. The process described involves collecting data from ThingsBoard using MQTT, and then using Apache Kafka to transfer the data to Spark Streaming for analysis. The data is cleansed and outliers are removed before being analyzed by Spark. This architecture is implemented in a smart health scenario where patients with respiratory issues are monitored. The sensors collect various measurements such as temperature, humidity, oxygen levels, CO2 levels, and NO levels in the patient's body.

Chen et al. [57] proposed a system that provides real-time monitoring data on water quality. This system is based on a wireless sensor network and developed using BIO (Bristol Is Open) to provide an experimental environment. The architecture they suggest consists of various modules such as data acquisition, power supply, data transmission, data storage, and redistribution. The data is transferred to data storage through WiFi and TCP/IP. To display the data through charts and graphs, they utilize a web application named Grafana. This monitoring system is efficient in measuring water quality parameters and displaying them, but it lacks alarm generation and data filtering before sending it to the database.

In [58] Rachmani et al. proposed a monitoring system to measure the PH and moisture in the soil. The measurements obtained from PH sensors and soil moisture sensors are sent to the client node, which comprises a LoRa transceiver, an Arduino microcontroller, a Wemos d1, and a power supply. The client node then transmits the data to the master unit through a LoRa protocol every 8 hours. The master unit, which is connected to the Internet, transfers the data to the cloud for monitoring. The sensor values are displayed to the user in three forms: the latest value, table, and chart. Additionally, a status icon is included to notify the user about the soil condition.

A monitoring and alarm system is proposed by Aarthi et al. [59] that is developed for drainage and waste management. This system can detect the blockage or high amount of toxic gases and generate alarms. The system is composed of various hardware and software components, including a DC power supply, an ultrasonic sensor, a gas sensor, GPS, a driver circuit, an IoT development kit, an Arduino microcontroller, and the Blynk app to control the devices over the internet using a graphical interface. The Blynk App is a versatile mobile application for IOS and Android that allows easy internet-based control of devices like Arduino and Raspberry Pi through a user-friendly dashboard. In their proposed system the data is received from an ultrasonic sensor, gas sensor and GPS and then it can measure drainage depth, toxic gas levels, and temperature in a remote field. By tracking these metrics in real time and sharing them online, unnecessary visits to manholes can be avoided, and only essential trips can be made. This system helps with monitoring of drainage and waste management and does provide alarms for such events as blockage or high levels of toxic gases.

The last related paper studied is proposed by Jha et al. [60] which is the most complete monitoring system we could find on this topic. The paper presents a framework that is a compilation of monitoring systems capable of monitoring all aspects of a smart city, ranging from air pollution to temperature and beyond. Their framework uses data analysis techniques and it can even predict how the data affects the smart city. In their framework for each smart city property, they created a separate monitoring system which includes air and noise monitoring and control system, a speed and web monitoring system (for vehicles), temperature and weather monitoring system (UV, wind,…), fire detection system, waste management

system, geographic information system. Their architecture also consists of a power supply, a display, a central control room and an alarm system. In their smart city model, all of the sensors and their corresponding systems are connected to a microcontroller unit(MCU) and the data are processed in that unit. Data analysis is done in this central unit to check whether the data are at dangerous levels and, if so, the system creates alarms, otherwise, the data is stored and displayed to the user.

The aforementioned works illustrate the range of uses of IoT within city environments and are focused on monitoring and observing smart city conditions. For example, in the Sakhardande et al. [52] approach they focused on hardware and each module of their approach had to have all hardware elements from sensors to the Arduino. In the work done in the city of Bandung [53], there is no means for triggering alarms based on changes in the stream of data. My City Dashboard [54] can present the statistics for each tile on the map or cluster, but the manager is unable to extract the information for a specific sensor or building in that cluster, therefore, they cannot locate the origin of the problem to fix the issue. Although Souly[55] is quite similar to our approach and is able to monitor the sensors, detect issues and even trigger alarms, it is created specifically for hotels and buildings and focuses on one building or hotel at a time. Even though in [56] the authors used the same IoT platform as ours, they directly fed it with their data and after that they did data cleaning and eliminating the outliers which could have been done before data reaching to the ThingsBoard.

Our approach to a model of the smart city is to ensure that the model can accommodate a wide range of assets and entities, particularly sensors and associated IoT platform, which is essential for a comprehensive smart city. This allows for a more holistic understanding of the city's overall functioning and performance. For example, we assume that there is (can be) data filtering between the devices and platforms, which helps in reducing the unnecessary transmission of data and thus conserves resources. We also assume that there is the flexibility to scale data display for specific sensors, buildings, groups of buildings, or the entire city. This ensures that the information presented is relevant and useful for decision-making and that those charged with managing the smart city have access to real-time monitoring dashboards

and the current city status, which allows them to make informed decisions based on accurate data.

Our approach stands out by not only introducing cutting-edge autonomic management but also by offering a seamless combination of this novel feature with administrator support, elevating it beyond traditional methods. In situations where the autonomic management alone may not fully address the complexities at hand, our system takes charge by promptly alerting administrators about any potentially hazardous conditions, contingent upon the corresponding policy being defined. This proactive and timely notification empowers administrators to respond swiftly and effectively to critical scenarios, mitigating risks with precision. This is seamlessly integrated into our existing infrastructure using the appropriate policy, eliminating the need for additional components. As a result, our approach not only ensures cost-effectiveness but also supports efficient operations. By leveraging our approach, administrators can receive real-time alerts regarding any dangers that may arise within the system. Our solution proactively detects and promptly communicates such situations to the relevant personnel, under the condition that the associated policy is in place. This timely notification system empowers administrators to swiftly address issues before they escalate into major problems, thereby mitigating potential risks and minimizing downtime.

Overall, our model aims to provide a comprehensive solution for monitoring and management of multiple assets and entities in smart city scenarios, while also being scalable and efficient.

2.1.3.2 Platforms for Smart Cities

Smart city platforms are crucial in providing the necessary support for managing the vast array of devices and systems that make up a smart city. These platforms provide a diverse array of vital capabilities necessary to facilitate the development and deployment of applications across a variety of smart cities, each possessing distinct functionalities.

One of the key capabilities of these platforms is cloud and fog support. This capability allows for the deployment of cloud and fog computing infrastructure, which can provide additional processing power and storage for smart city applications. This can help to improve the performance of smart city systems by reducing latency and improving data processing speed.

Another important capability of smart city platforms is big data management. These platforms are equipped to handle the large amounts of data generated by smart city devices and applications. This data can be analyzed and processed to provide valuable insights and inform decision-making. Stream processing is also an essential capability provided by these platforms. It allows for real-time data processing and analysis, which is crucial for applications that require immediate action based on real-time data, such as traffic management systems. Dynamic network compatibility is another key feature of smart city platforms. This capability enables the integration of different network protocols and technologies, which is essential for ensuring seamless connectivity and communication between different devices and systems. Some of the most popular Internet of Things (IoT) platforms for building smart city applications available in the market include IBM Watson IoT, Microsoft Azure IoT, Google Cloud IoT, and AWS IoT. These platforms offer a suite of tools, services, and infrastructure to build, deploy, and manage IoT solutions at scale. They enable organizations to connect and communicate with a vast array of IoT devices, collect and analyze data, and derive actionable insights. In the context of smart city scenarios, these IoT platforms can be utilized to create and manage various applications and services that improve urban infrastructure, enhance resource management, and enhance the overall quality of life.

Smart city platforms play a critical role in supporting smart city scenarios by providing essential capabilities such as cloud and fog support, big data management, stream processing, and dynamic network compatibility. The availability of a wide range of platforms in the market ensures that there is a platform that can meet the needs of different smart city applications. In this section, we will have a look at some of the smart city management platforms and their key features.

SmartCityWare [61] is a service-oriented middleware that focuses on smart cities. The services in this platform are categorized into core services such as broker services, invocation services, location-based services, and security services, as well as environmental services that provide access to services provided by clouds, fogs, or devices. SmartCityWare uses Cloud of Things (CoT) to improve computation, optimization, and decision making along with Fog Computing to decrease latency, provide location-wise services, a better quality of service,

support for distributed applications, and so on. Although this integration provides both local and global monitoring, configuration, optimization to the smart city applications, it can be challenging for the developers.

GAMBAS [62,63] middleware also focuses on smart city applications and provides a Software Development Kit (SDK) and a runtime system that can be used to develop such applications. This middleware can offer efficient data acquisition, privacy-preserving data distribution and interoperable data integration. GAMBAS architecture consists of three components: Android runtime, J2SE runtime and Distributed registry.

CityPulse [19] is also a framework for big data processing, analytics and interpretation in smart cities that allows developers to create services and applications for citizens. This framework is able to capture, store and process the stream of data in real-time and provide the city condition at any moment along with the events and happenings in the city. CityPulse is composed of several components. The first set of components is categorized under large-scale data stream processing modules which allow data source interaction, data stream discovery and analysis. These components are data wrappers, resource management, data aggregation and federation, event detection, quality monitoring, fault recovery, geospatial database, and city dashboards. The next set of components is contextual filtering, event-based user-centric decision support and technical adaptation which fall under the adaptive real-time decision support module and can provide suggestions based on the state of the city. This framework is more advanced than the two smart city frameworks mentioned before, but it has not been updated since 2016 [64].

2.1.4 Autonomic Computing

Autonomic computing is defined by IBM in 2005 [65] as the way that the computer systems respond to a change in the complex IT environment like healing, adaptation, or protection. Based on this guide, four attributes should be included in the system to have an autonomic computing system:

- Self-configuration: The IT environment should configure itself to adapt to the changes.

- Self-healing: The IT environment should detect problematic operations and recover from them using the proper corrective actions.
- Self-optimizing: The IT environment should be able to improve resource utilization and performance of the system.
- Self-protecting: The IT environment should protect itself against attackers and vulnerabilities by taking proper actions.

The components in an autonomic environment can work together and manage themselves and each other. A control loop as shown in Fig. 2.1 is a significant part of the autonomic architecture that has four main parts with access to shared knowledge. These four parts are monitor, analyze, plan, and execute forming the MAPE model.

- Monitor: This part is responsible for collection, aggregation, filtering and reporting
- Analyze: The analyze part should analyze the monitored data and model the situation
- Plan: This part uses the policies and analyzed results to select the best action to adapt to the change
- Execute: In this part the plans should be executed to achieve the goal

Autonomic computing is beneficial to business systems and results in operational cost reduction, lower failure rate and response time, and increased security.

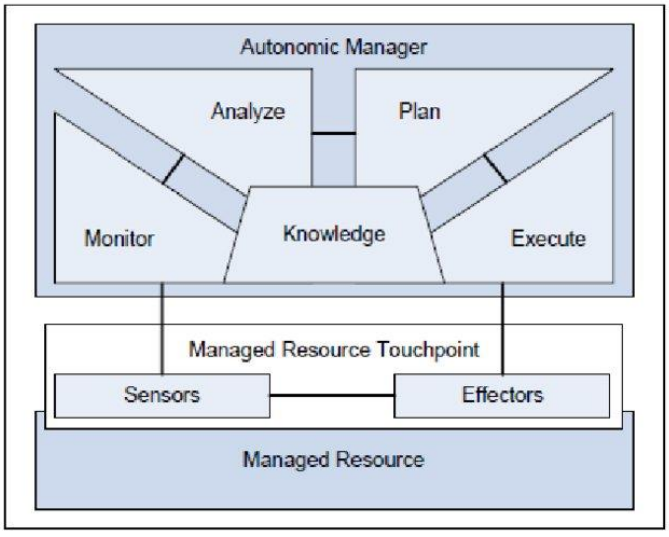


Figure 2.1 MAPE-K Control Loop [65]

Autonomic computing refers to systems that consist of autonomic elements that can manage themselves to accomplish the high-level goal set by the administrator [66]. This notion first

was inspired by the autonomic nervous system and introduced by IBM. Autonomic computing is defined as a large number of autonomous components that are formed in a hierarchy and are able to self-govern and interact with each other. The main principle of autonomic computing is self-management which means that the system can manage, maintain and adjust itself to the dynamic world and there is no need for a human to monitor and manage it. This type of system can monitor and optimize its status, detect errors, check for upgrades and configure itself in real-time. In this paper [66], it is mentioned that the autonomic system is a distributed and service-oriented infrastructure that is made by autonomic elements that interact with each other and manage their internal behaviour based on the rules that are set by the system administrator or even other elements. The autonomic elements' tasks can correspond to their level in the hierarchy, for example in the higher levels the elements have more flexibility and are more dynamic. There can also exist an autonomic manager in the system that is responsible for monitoring and analyzing the elements and the entire environment and creating action plans for the autonomic system. Finally, in this paper, some of the engineering and scientific challenges of autonomic computing have been indicated.

Sterritt et al. [67] also pointed out the origin of autonomic computing and provides an introduction to it. Its definition is very close to the above-mentioned one and like most of the papers in the realm of autonomic computing believes that an autonomic system is a system that is autonomic and self-managing environment and can hide the complexities in the system from the end-user. This paper also introduces some self-* features that are crucial to autonomic systems such as self-governing, self-adaptation, self-organization, self-optimization, self-configuration, self-diagnosis of faults, self-protection, self-healing, self-recovery, and autonomy. The authors also considered the relationship between artificial intelligence and automaticity. AI techniques such as soft computing techniques, machine learning approaches and others, can be beneficial in helping to provide an autonomic environment. For example, Bayesian networks can be used for selecting the suitable autonomic algorithm or Markov decision process can be used for failure remediation, etc.

Parashar et al [68] provide an introduction to autonomic computing and its challenges. Similar to other works that have been done in this area, it reiterates that the idea of autonomic

computing comes from the human nervous system. However, the authors found more similarities between autonomic computing and the nervous system and talked about a dynamic equilibrium and the fact that the system reacts to the change in the environment to maintain that equilibrium. In addition to that, they introduced Ashby's ultra-stable system [68] in which some variables are defined that should be kept in a specific physiological threshold to guarantee the adaptiveness of the system. As a result, they believe that in the autonomic computing paradigm there should be a procedure that maintains this stable equilibrium in response to the changing environment. Some of the actions that can be taken to keep the stability of the system can be self-protection, self-recovery from failures, reconfiguration, the attempt to keep the performance of the system and self-optimization. Finally, some of the challenges are presented such as conceptual challenges, architecture challenges, middleware challenges and application challenges.

Calinescu published a paper [69] related to general-purpose autonomic computing in which three criteria were identified that should be fulfilled for the generality of the autonomic computing system. These criteria are:

- The autonomic computing framework should support heterogeneous ICT components such as software, hardware, etc.
- It should provide self-* features and support for autonomic policies.
- The framework should decrease the cost and effort of creating the autonomic system by enabling the developers to reuse the components and provide standards for different elements of the system and support modular development.

In this paper [69], an architecture for general purpose autonomic systems is proposed which consists of a reconfigurable policy engine. This component is the main component of the system architecture, and it can manage resources and define the objectives that are defined by the user. To ensure the third criterion, in the architecture some adaptors are developed and utilized by using the interfaces for sensors and actuators. The policy engine includes components such as a runtime code generator, manageability adaptors proxies, high-level manageability adaptors, a scheduler, resource discovery, machine learning modules, and a probabilistic model checker.

The focus of this section was on autonomic computing, its definitions and its characteristics. The reason for studying autonomic computing is that for implementing the autonomous management system we should be aware of autonomic computing, its properties and its essential components. We found out that a system should include self-* attributes in order to have an autonomic computing feature. Some self-* attributes include but are not limited to self-configuration [65] [67], self-healing [65] [67], self-optimization [65] [67] [68], self-protection [65] [67] [68], self-management [66] [67], self-governing [66] [67], self-adaptation [67], self-organization [67], self-diagnosis [67]. self-recovery [67] [68], and so on. In the studied papers, some methods have been utilized in order to have such a system for example in [65] a control loop is the main feature of the system which is based on the MAPE model. The authors in [66] introduced autonomic elements and autonomic managers. [67] benefited from AI techniques to bring automaticity into the system. In [68] the technique was based on a dynamic equilibrium inspired by the nervous system. Finally, in [69] their proposed architecture was based on a policy engine.

2.2 Literature Review

This section delves into a comprehensive examination of previous research focusing on autonomous management. Our exploration concludes with a thorough analysis of each paper, shedding light on the specific aspects addressed by each, and discussing any gaps or shortcomings in comparison to our proposed solution. By navigating through the current state of research in this domain, we aim to provide a detailed understanding of the existing literature landscape, paving the way for a more insightful evaluation of our proposed approach.

2.2.1 Autonomous Management

Autonomous management is a critical aspect of modern systems, especially in the context of complex and large-scale systems such as the Internet of Things (IoT) and smart cities. It refers to the ability of a system to manage itself without the need for constant human intervention. This section will delve into the concept of autonomous management, its key features, its applications in various domains and the approaches that have been proposed. We will also explore the challenges and opportunities associated with implementing autonomous management in complex systems. In Section 2.2.2, we will engage in a comprehensive

analysis of the papers centered around autonomous management. We will delve into the constraints and limitations of each study, highlighting the aspects they might have overlooked. We are seeking specific attributes in the papers under consideration. Initially, it is important to conduct a thorough assessment of the proposed solutions, encompassing prototype development, experimentation, and the presentation of case studies. A meticulous explanation of their structural intricacies and the technologies utilized is essential for a comprehensive evaluation. Moreover, a robust solution for autonomous smart city management should not only monitor and control the smart city environment but also its infrastructure. Operational performance is another focal point for a desirable solution. In the context of policy-based management, a desirable characteristic is the ability to seamlessly incorporate new rules and policies. Moreover, the solution's generality and adaptability to accommodate diverse requirements and network configurations are other pivotal considerations. This comparative evaluation will underscore the distinctive contributions of our research.

In [70] Gurgen et al. introduced the characteristics of autonomic systems, and then they provided suggestions for self-aware cyber-physical systems for smart cities. The common properties of autonomic cyber-physical systems and a brief description for each of them are described in the following:

- Self-adaptation means that when a change happens, the system should adjust itself to comply with the main objective.
- Self-organization means that the system should be able to organize itself dynamically when nodes are being added or removed from the smart city network.
- Self-optimization is when the system is able to manage the usage of its limited resources in an optimized way.
- Self-configuration indicates when the system should perform real-time configuration of the devices automatically.
- Self-protection is when the system can protect itself against attacks happening in the smart city without sacrificing the quality of service and experience.
- Self-healing means that the system should monitor itself and fix the detected problems as fast as possible.
- Self-description is another property of the system that belongs to each node in the system. Because of communication in the network, each node in the system should provide some information about itself to other objects.
- Self-discovery is that the devices and even services in the system should be discovered automatically by the system.

- Self-energy-supplying means that the power supply of the entire system should be self-harvesting.

This paper suggested using the MAPE-K (Monitor- Analyze- Plan- Execute and Knowledge) model which is a well-known model for autonomous systems along with SOA and cloud computing to create a modular, reconfigurable, and extensible self-aware system. The crucial services provided by their self-aware middleware are data collection and processing, the composition of sensors and actuators, device management, and autonomic management. In the end, some suggestions are provided for self-manageable systems.

Braten et al. [71] proposed a generalized cognitive model for autonomous IoT device management that reduces human intervention. This approach introduces a model that consists of two managers: the device manager, which is in charge of analyzing and planning the devices using the digital twin concept, and the system manager, responsible for assessing the system environment and determining how the external conditions influence the system and the devices. Each manager has its own procedural and declarative knowledge subcomponents. Moreover, the system contains control and adaptation loops. The adaptive loop follows the MAPE-K pattern, and it can take two paths based on the change in the environment or on the devices. There are two learning loops for each system manager and device manager respectively, and the last loop, the autonomic loop, is responsible for controlling the device in the short term. There are also explicit triggers in their model and their task is to control the data flow and behaviour of the system in accordance with decisions that are made locally. Using digital twin, this model ensures that the actions and adaptive instructions are synchronized because otherwise the manager's interpretation of the effect of the action might be incorrect and learning processes might suffer. This work focuses only on the environmental aspect of the IoT network and management of the IoT devices and they do not consider the operational aspects of the infrastructure.

Kyriazis et al. [72] discussed the challenges and enablers for creating smarter, more reliable, and autonomous systems within the context of the Internet of Things (IoT). Their proposed conceptual architecture aims to enhance the sustainability of IoT applications by utilizing a large number of heterogeneous device platforms efficiently. The architecture incorporates

cross-platform channels for data, information, things, and decentralized management as shown in Fig. 2.2. These channels enable the development of an environment for IoT applications and facilitate the acquisition and analysis of situational knowledge to make things aware of conditions and events affecting IoT systems' behavior. For each channel, its tasks are represented in Fig. 2.2. The devices in the proposed architecture have the capability to share their data. Then in the information channel knowledge is derived from the data and leads the devices to be more autonomous. The decentralized channel in the proposed architecture utilizes technologies for efficient management and coordination of a large number of IoT devices. Rich metadata structures capture the "social behavior" of things, facilitating decentralized management of networks of things. The management mechanisms are described as "enhanced and autonomous," and their major importance lies in their ability to adapt to real-world situations. By continuously analyzing raw data, the management mechanisms can provide a snapshot of the network of things' behavior and state at any given time. This analysis triggers actions concerning resource and data management, allowing for autonomous decision-making and efficient operation of the IoT ecosystem. While the proposed architecture showcases ideas such as decentralized management, enriched metadata structures, and autonomous decision-making, the absence of practical realization prevents it from being validated and applied in real IoT environments.

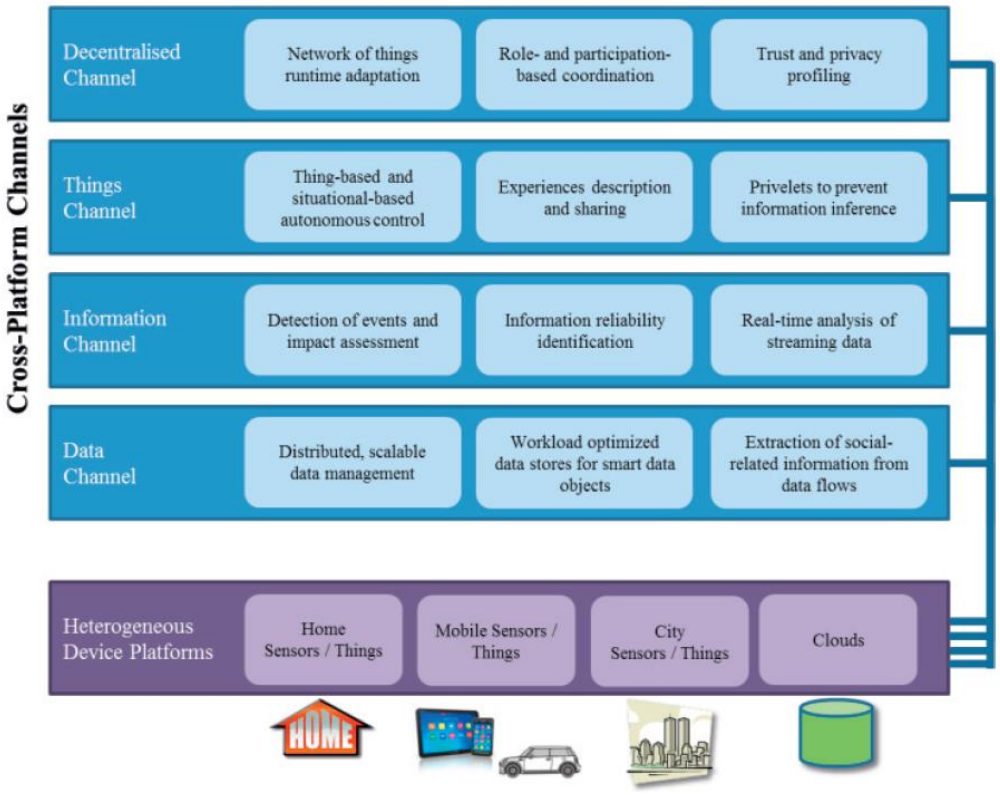


Figure 2.2 Cross-Platform Channels [72]

In [73] Gurgen et al. proposed an approach based on ECA rules for autonomic management of sensing devices. According to this paper, smart devices are autonomous if they are self-discovered, self-configured and self-healed. In this paper, a self-manageable autonomic platform is introduced which is based on the Event-Condition-Action paradigm. In the ECA paradigm, an action will occur when a specific event with a determined condition happens. An event is defined by four fields: type, content, element Id and timestamp. Also, management actions can be done to configure, install software or even perform a diagnosis using SET, GET and ACT.

Another work that has been done in the field of autonomous management is mentioned by Sharrock et al. [74]. This paper also introduces a middleware that, like the paper by Gurgen et al. [73] uses ECA rules to provide the management of heterogeneous devices autonomously. This management offers remote deployment of software, performance monitoring, and dynamic configuration of the sensing devices through high-level policies. To create this

middleware, they borrowed encapsulation mechanisms from TUNe [75] and applied those mechanisms to their management middleware, XSStreaMWare. TUNe is an autonomic management system that uses the Fractal component model which is a type of component model that supports modularity for a system and encapsulates software using wrappers and provides server and client interfaces to support incoming and outgoing method calls [75]. The XSStreaMWare middleware benefits from an ECA engine that upon happening an event with certain conditions, the defined action will take place. To define the ECA rules, they developed a Sensor Management Modeling Language (SMML). XSStreaMWare has a service-oriented distributed architecture, and its approach is dividing the environment into several regions and managing the regions by using a gateway that hosts adapters. Because the devices join or leave the network dynamically, upon detecting a new device this middleware checks the firmware version on the device and if it needs to be updated, the middleware removes the current version and installs the latest one. In addition to the software/firmware version, this middleware is able to reconfigure the device or even monitor and manage the overall performance. As mentioned, the ECA rules are defined in a graphical language called SMML (Sensor Management Modeling Language).

A multi-agent-based autonomic network management architecture is proposed by Arzo et al. [76] that uses mathematical modelling. This paper begins by summarizing the challenges of network management automation including system complexity, dynamicity, heterogeneity, and data volume. Paying attention to the network management cycle is important in order to provide automation. This cycle is composed of several steps: environment measurement, decision making, planning action strategy, and verifying it, and at last, action execution. Automated decision-making task contains six subtasks of data analytics, decision generation, organization, verification, execution, and monitoring system behaviour. Their management architecture, called MANA-NMS, divides the complex system into several reusable atomic modules that can perform each task autonomously in interaction with the environment and other agents. Their proposed architecture includes multiple autonomous agents that are based on intelligent algorithms like ML or DL. The management function can be done in three functionality levels: the physical layer, the device layer and the network layer. This approach

can effectively monitor and manage the network, but it does not consider the management of components of the broader smart city infrastructure which our research looks to address.

Mezghani et al. [77] explores autonomic coordination of IoT device management that detects the dependencies between the isolated device management (DM) platforms and coordinates them, meaning that when an operation is going to be executed on the devices, this middleware autonomously organizes that execution. Their approach benefits from a knowledge component and has two phases: during the first phase, the integration phase, it accumulates the knowledge about DM servers, managed devices and device dependencies and in the second phase, the coordination phase, using the accumulated knowledge it performs several tasks such as planning the order of operations based on the coordination rules and arranging their execution. This coordination has several levels of complexity from static to dynamic. The authors provide details about the architecture and implementation of the first level coordination complexity which is static time-window coordination.

The work by Ayeb et al. [78] focuses mainly on device management based on the coordination of autonomic loops for target identification, load, and error-aware device management for the IoT. Device management is described as consisting of four major operations as maintenance (firmware updates), provisioning, monitoring, and troubleshooting of the heterogenous devices that should be done remotely, constantly and without physical intervention. Heterogeneity, dynamicity, and scalability are the challenges of the IoT DM that they tried to address in their architecture. This paper mentions that after applying the DM operations, the devices should be monitored constantly to ensure that they continue operating as expected and the operation was successful. Their proposed architecture is a composition of three autonomic loops that are connected and can interact with each other: 1) operation generation and target identification, 2) decomposition enforcement and tracking, and 3) speed regulation. Their approach was interesting for the following reasons: i) specific devices can be identified and the operations can be performed on only those devices, ii) there is a monitoring component that observes notifications on firmware and configurations, device states, and some metrics about the infrastructure. Monitoring infrastructure metrics is done to detect QoS variations, diagnose warnings, and mitigate errors. The Decomposition, Enforcement and Tracking loop

monitors and adjusts device operations using various metrics, including average CPU usage and RAM load.

Kosińska et al. [79] proposed a model-driven autonomic management framework for cloud-native applications called AMoCNA using high-level policies. An abstract view of the model is shown in Fig. 2.3. This model helps developers to understand which components are essential for a cloud environment. In their approach, the management process is done in the same place as the CNApps namely Cloud-native Execution Environment and this environment is controlled by Cloud-native Autonomic Manager. Requirements considered in this framework include:

- The solution should be goal driven.
- To guarantee application health and optimization, the framework should log metrics at every layer and metrics should get aggregated.
- The architecture should include monitoring.
- The QoS level should be kept as expected using the observable information of all execution components.
- Autonomous management is achieved using high-level goals to prevent human intervention.
- The feedback system is based on MAPE-K.
- The solution should be extensible and take the environmental dynamicity into consideration.

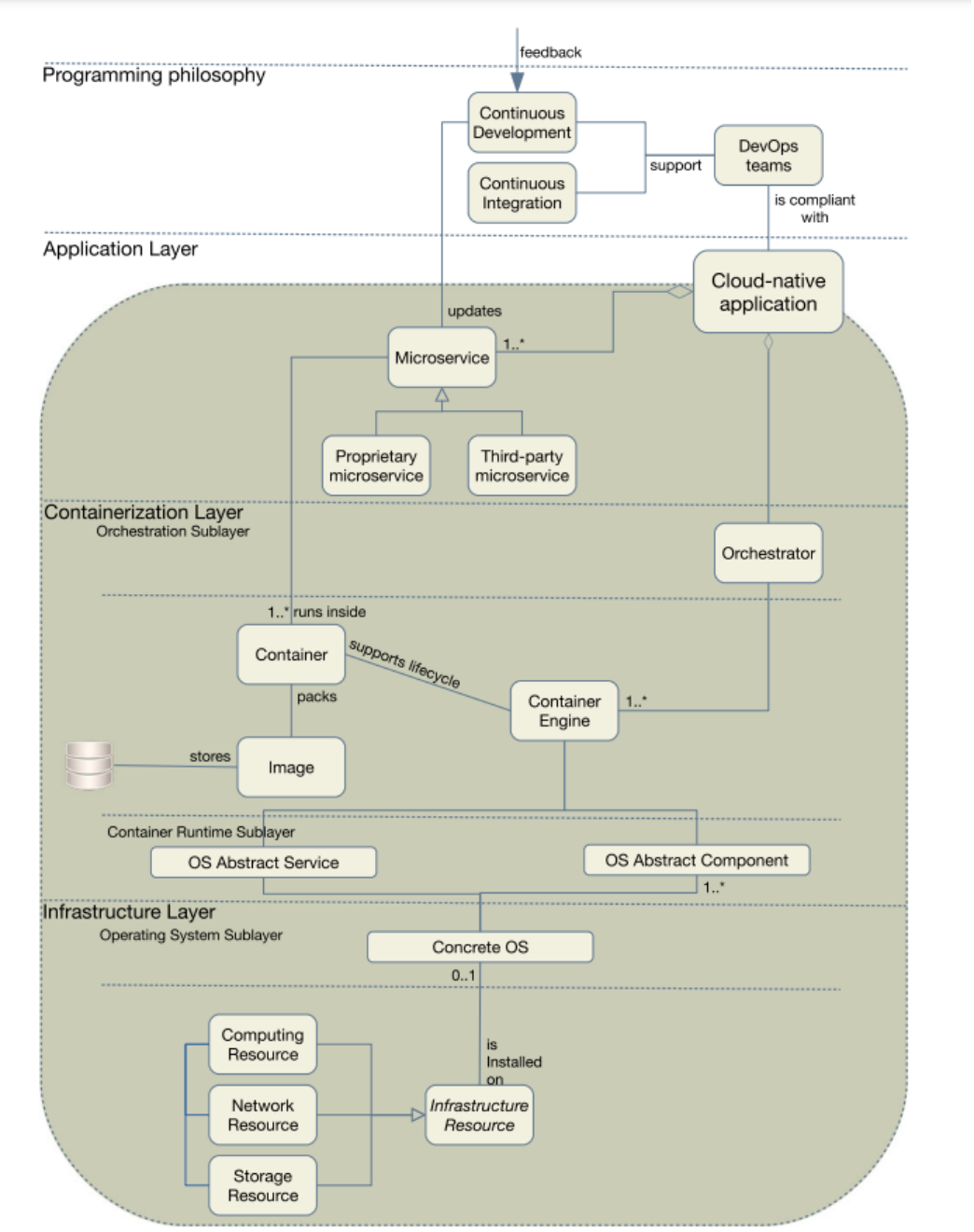


Figure 2.3 Abstract Model of a Cloud-native Application Execution Environment [79].

Some components are important in this solution. For example, the Declarative management policy component refers to a high-level specification or set of rules based on the event-condition-action concept that dictates the desired behavior and actions to be taken in managing a cloud-native environment. It provides a mean to define the expected outcomes and behaviors

without specifying the exact implementation details. The declarative management policy is a key component of the autonomic management framework for cloud-native applications (AMoCNA) and allows for policy-driven management and enables automation and autonomy in the management process. It guides the actions and decisions of the Cloud-native Autonomic Manager, which enforces the policies and carries out the necessary management tasks. The AMoCNA model has five logical layers of components: Instrumentation, Observation, Management, Inference and Control. This framework has two types of microservices: management policy microservice that is responsible for managing the whole execution environment, and an autonomic element microservice. AMoCNA utilizes a rule engine for enforcing management policies in a Cloud-native context. However, specific details about the policies, such as their implementation and examples, are not explicitly mentioned. The text suggests that policies are defined using rules, and it can be inferred that policies may involve conditions and actions based on the collected metrics, allowing for dynamic management decisions. As an experiment, they defined two different policies to adjust the requests for CPU and found out that as the number of rules increases, the delay will increase as well.

Having said that, this approach is the closest one to our research, but it lacks explicit consideration of smart cities and does not specifically mention or address smart cities as a domain of application for the proposed autonomic management framework. The focus of the paper is on the autonomic management of cloud-native applications, and the discussion revolves around cloud-native environments and their management requirements. Therefore, the paper does not explore the application of the framework in the context of smart cities. Therefore, they do not monitor and manage both the infrastructure performance and the sensor measurements.

Decision-making support for an autonomous software-defined network orchestrator is presented by Saadon et al. [80]. In this paper the authors focused on autonomous SDN management, and they proposed a mathematical function to support decision-making at the orchestrator level. Using this function, the orchestrator can decide which rule should be selected to ensure that the resources are allocated in an optimized way. Their proposed method starts by determining the applicable rules and then the mathematical function estimates the

rules and assigns a weight to each rule. Finally, the rule with the highest score will be selected by the orchestrator. Their focus is on software defined networks, and they only take applications and the rules into consideration. Also, using mathematical modelling despite linearity can have limitations and add unnecessary complexity.

Lam et al. [81] proposed an autonomic approach based on the MAPE-K model for dynamic orchestration and configuration services in Industrial IoT. In this research, the control system is under real-time monitoring and adaptation takes place based on semantic policy to automate the orchestration. In the context of the Autonomic Adaptation System, semantic policies are expressed using Semantic Web technologies, such as the Semantic Web Rule Language (SWRL), SPARQL queries and the Resource Description Framework (RDF). Orchestration policies are expressed using the Semantic Web Rule Language (SWRL) and the monitoring data is stored in the central Knowledge Base (KB) using the Jena RDF (Resource Description Framework) Triplestore3. These technologies provide a standardized and expressive way to represent knowledge, making it easier to reason over the information and make informed decisions regarding system adaptation. As a practical use case, they used their approach in supply chain management to automate the monitoring of the storage tanks and supplying the raw material. This approach still depends heavily on having an administrator for management and is not employed in smart city contexts.

Sampaio et al. [82] proposed a system called the Central IoT (CIoT) system, which aims to optimize power consumption in Internet of Things (IoT) and Fog Computing setups. The system utilizes advanced orchestration mechanisms to manage dynamic duty cycles, resulting in significant energy savings. It operates autonomously without requiring human intervention and can adjust power cycles based on contextual information such as environmental conditions, user behavior, regulations on energy utilization, and network resources. By leveraging IoT devices and Fog Computing, the CIoT system intelligently controls and optimizes power consumption in various scenarios, including smart homes, smart cities, smart agriculture, intelligent fire alarms, and intelligent traffic lights. The system's autonomic nature is attributed to its ability to function and optimize power consumption without human intervention, relying on advanced orchestration mechanisms provided by Fog Computing.

Although the specific policies used in the CIoT system are not mentioned, it can be inferred that policy-based mechanisms guide the system's operation and decision-making process to regulate energy utilization and optimize power consumption in an autonomous manner.

Nalinaksh et al. [83] proposed autonomic internet of things ecosystems focusing on the application of autonomic computing in IoT deployments. It outlines four real-world pilots from the ASSIST-IoT project: port automation, smart worker protection, car engine monitoring, and car exterior monitoring. The paper emphasizes the lack of ready-to-use tools for designing model-based architectural self-adaptation in IoT ecosystems. The autonomy-related requirements derived from the ASSIST-IoT project pilots include self-configuration, self-healing, self-optimization, and self-protection mechanisms across all four pilots. These requirements aim to enable next-generation IoT ecosystems with semi-autonomic behaviors.

The paper by de Sousa et al. [84] introduces CLARA, a Closed Loop-based Zero-touch Network Management Framework for 5G systems. CLARA utilizes technologies like SDN, NFV, Network Slicing, and Intent-based Networking, along with policy-based closed control loops (CCL) to automate network and service management. It enables self-x properties and fulfills user requirements in a multi-domain environment. The framework leverages policies derived from user intent and implements them through a knowledge base, enabling flexibility and adaptation. CLARA's contributions include a modular CCL platform, translation of service intents into monitoring models, and prototype implementation. The paper emphasizes the need for novel approaches in network management to meet modern network and service demands, and CLARA aims to provide a comprehensive solution to achieve efficient network management and service provisioning.

Kosinska et al. [85] introduce an experimental evaluation of a rule-based autonomic computing management framework for cloud-native applications. The focus is on assessing the effectiveness of this approach in governing system behavior and achieving compliance with CI/CD (Continuous Integration /Continuous Delivery/Deployment) objectives. The paper proposes a methodology for evaluating complex cloud-native environments and presents two categories of experiments. The first category evaluates the impact of dynamic resource adjustment on the system, emphasizing the importance of observability in

understanding the application state and the effectiveness of policy-based management. The second category assesses the rule-based approach's influence on autonomic management, demonstrating its accuracy even in scenarios with frequent modifications. The paper discusses the concept of a rule engine for enforcing management policies and highlights the benefits of the proposed MRE-K (Monitoring, Rule Engine, Execute, and Knowledge) loop concept. It also compares the declarative management policy approach to Kubernetes' available options.

Villela Zavala et al. [86] proposed an Autonomic IoT framework to enable full autonomic behavior in IoT systems. It tackles challenges related to scalability, diversity of application domains, device heterogeneity, large device volumes, and ubiquity. The framework incorporates the Autonomic Control Loop (ACL) MAPE-K, consisting of four components: monitor, analyzer & planner, executor, and knowledge. The proposed framework does not require changes to the existing IoT architecture but adapts the system to specific IoT domains. It uses Utility Theory to achieve self-configuration and self-management, allowing the system to autonomously adapt and configure itself based on current conditions and requirements. The framework is validated through an experimental testbed in urban agriculture, specifically using hydroponic and aeroponic systems to grow maize crops. The experiments demonstrate that the proposed framework can manage the growth cycles of crops without human intervention, optimizing plant phenology and improving crop yields. The paper highlights the adaptability and optimization capabilities of the framework, making it suitable for various application domains. However, potential limitations include scalability challenges, implementation complexity, and resource constraints. In summary, the paper presents a promising Autonomic IoT framework that combines control principles and Utility Theory for self-configuration and self-management in IoT systems. The experimental results in urban agriculture show its potential benefits, but further research is needed to address potential limitations and ensure practicality in real-world IoT deployments.

The paper by Riker et al. [87] introduces AGREEN, an autonomous solution for the autonomic management of group communication in Dense Internet of Things (DIoT) applications. AGREEN aims to achieve self-adaptable communication in DIoT environments by addressing challenges such as message loss, congestion, and energy consumption. It utilizes a group-

oriented approach, creating monitoring groups of IoT devices with similar characteristics to facilitate efficient communication management. The fuzzy decision system in AGREEN considers various factors to dynamically adjust communication settings, including traffic loss, sensing relevance, energy harvested, and the number of monitoring nodes with renewable and non-renewable energy. AGREEN addresses energy consumption by utilizing a fuzzy logic controller to autonomously detect low-performance situations and make automated decisions regarding the number of group members, communication interval, and confirmable message rate. The paper identifies gaps in the existing literature regarding autonomic control of communication reliability, traffic production, and energy balancing in DIoT networks with heterogeneous energy sources. While AGREEN shows promise for achieving self-adaptable communication, the paper acknowledges limitations, such as the need for customization in different DIoT environments, and scalability challenges for large-scale networks. Overall, AGREEN presents an energy-efficient and reliable solution for DIoT scenarios, but further research and improvements are needed to address the identified limitations and make it more suitable for diverse IoT environments.

Singh et al. [88] proposed autonomic resource management in a cloud-based and distributed system environment. The main objective of autonomic computing is to achieve self-management capabilities, allowing the system to handle computing resources dynamically while concealing complexities from users. The approach focuses on four key aspects: self-configuration, self-healing, self-optimization, and self-security. The paper proposes an autonomic engine for cloud management that embraces heterogeneous and dynamic cloud architectures, predicting IT system needs and reducing human involvement in issue resolution. Contributions of autonomic computing include self-configuring modules that adapt to changing conditions, self-healing components detecting and correcting defects, self-optimizing modules adjusting services to meet user needs, and self-protecting features defending against attacks. However, the paper lacks detailed implementation information and comprehensive experimental analysis, while also inadequately addressing challenges. The conclusion emphasizes the need for a transformative shift in software system development to accommodate autonomous computation, calling for further research, and interdisciplinary work to address the challenges and enhance the practical relevance of autonomic computing.

Shukla et al. [89] explored the concept of autonomic cloud resource management within the context of Industry 4.0, which refers to the fourth industrial revolution characterized by automation, machine-to-machine and human-to-machine communication, and digitization in various industries. The main focus lies in examining how autonomic cloud computing can significantly enhance resource management and improve the efficiency of cloud services. The concept of autonomic computing is introduced, emphasizing its pivotal role in self-managing computer systems through adaptive technologies, thereby reducing the dependence on human intervention. Autonomic cloud computing is discussed as a means of leveraging cloud resources and services to meet user demands in a flexible and efficient manner. The paper delves into the challenges of managing cloud resources autonomously, which include the critical need for scalability, energy efficiency, and cost optimization. It introduces the notion of a resource pool and self-configurability to tackle these challenges effectively. A significant portion of the paper is dedicated to presenting a comprehensive architecture and methodology for autonomic cloud resource management. This methodology focuses on key aspects such as job execution, fault tolerance, and optimization processes. The paper emphasizes the essential role of autonomic computing in effectively managing cloud resources in this context. Various components and entities involved in the proposed architecture are described in detail. These include users, larger and smaller datasets, self-configurable nodes, fault detection mechanisms, and the resource pool. The paper concludes with a strong emphasis on the significance of autonomic cloud resource management in Industry 4.0. It highlights the numerous benefits of the proposed architecture, particularly in terms of resource utilization, user experience, and overall efficiency. Additionally, it acknowledges the need for further research and development in this evolving field. Overall, the paper underscores the immense potential of autonomic cloud resource management in addressing the challenges of resource allocation, scalability, and energy efficiency in cloud computing within the dynamic landscape of Industry 4.0.

Mangla et al. [90] proposed a framework and architecture for autonomic resource management in cloud computing environments, with a focus on integrating fog computing and IoT. The framework aims to enable intelligent decision-making and efficient resource allocation through a control loop based on MAPE in response to the environment to achieve self-

management. The proposed architecture introduces a three-layered approach: cloud layer, fog layer, and IoT layer. The fog layer, placed near end devices, acts as an intermediary between the cloud and IoT to optimize latency and address data processing challenges. It consists of a fog master and fog slaves, where the fog master performs IoT services and manages the fog slaves. Tasks are forwarded to the fog master from the IoT layer, and it decides whether to handle the task itself, delegate it to fog slaves, or send it to the cloud layer based on network parameters and resource requirements. The fog master serves as an autonomic manager, conducting functions such as monitoring, analyzing complex situations, planning actions, and executing them dynamically. Implementing autonomic computing in the fog layer reduces the burden on the cloud layer and enhances overall performance. The proposed architecture emphasizes efficiency in processing and energy consumption by leveraging the computational resources of fog nodes and the extended capabilities of the fog master. By integrating fog computing and autonomic computing principles, the architecture aims to optimize the processing of IoT data and overcome cloud computing limitations related to data volume and complexity.

Patibandla et al. [91] investigated the concept of autonomic computing in cloud computing using architecture adoption models and its potential benefits for organizations, emphasizing improved resource utilization, cost optimization, data redundancy, and dynamic security modifications. It presents a case study focusing on the application of autonomous cloud management in the context of dengue fever prediction, a critical public health issue in tropical regions. The study utilizes the Cloudbus Workflow Engine, designed for grid-oriented workflow management, to automate the workflow process and optimize resource allocation. The workflow model is tailored to process and analyze spatial-temporal data related to dengue fever, aiming to achieve accurate and timely predictions for effective control. The methodology for performance evaluation involves several steps. A theoretical testbed is created, consisting of a hybrid cluster with multiple processors and memory, along with virtual machines deployed in a local network. The dengue fever prediction program utilizes recorded dengue cases and environmental evidence from a specific time period. The workflow design incorporates iterative scheduling and optimization, with tasks executed iteratively, adjusting resource allocation based on input data. The iterative scheduling algorithm actively adjusts

resource allocation and scheduling based on estimated execution time and cost. The optimization process follows a series of steps to reduce costs and consider resource constraints, including launching cloud resources, applying a greedy algorithm, and analyzing public cloud resources' cost-effectiveness. The performance evaluation demonstrates the effectiveness of the automated iterative optimization function, reducing runtime by 48% and public cloud usage expenses by 70% compared to a selfish approach. These results highlight the significance of adopting autonomous cloud management in various sectors beyond dengue fever prediction, such as eHealth, e-Science, e-Government, and e-commerce. The paper concludes by underlining the importance of autonomous software management and its potential to address cloud infrastructure management challenges.

Zhou et al. [92] introduced SeaNet (Semantic Enabled Autonomic Management of Software Defined Networks), a graph-driven approach for autonomic management of software-defined networks, which harnesses the power of artificial intelligence for telecommunication network management. It begins by discussing the historical development of AI in this field and highlights the limitations of existing proposals that lack practical implementations and evaluations. To address these shortcomings, the authors introduce SeaNet, an architecture based on knowledge graphs and ontologies for autonomic network management. SeaNet's architecture comprises a knowledge graph constructor, a SPARQL engine, and a network management API. The knowledge base generator harmonizes unstructured data from various network sources into RDF triples using ontological concepts, enabling efficient reasoning through first-order logic formulas. To validate the practicality of SeaNet, extensive evaluations are conducted on computer networks and Wi-Fi networks using Mininet and Mininet-WiFi. The knowledge base generator is evaluated in terms of response time, showing a linear relationship with network scale. The overhead for generating knowledge bases is deemed acceptable for various network topologies. The Network Management API is rigorously evaluated, comparing SeaNet's methods to a leading network management controller, Ryu. SeaNet's "connectAll" method outperforms Ryu's application, especially in large networks, and exhibits superior code complexity, execution time, readability, and reusability. Additionally, the scalability of SeaNet is demonstrated through RDF reasoning. The efficiency of RDF-based reasoning remains consistent and unaffected by the scale of the

knowledge base. Overall, the evaluations confirm that SeaNet is a practical and efficient AI-driven solution for autonomic SDN management. By leveraging knowledge graphs and ontologies, SeaNet provides a technology-independent approach, simplifying complex network tasks while achieving high-performance results in various network scenarios.

Lin et al. [93] proposed autonomic security management for IoT smart spaces. A "smart space" can be described as an environment that integrates embedded sensors and devices. The primary objective of this research is to enhance the security of smart spaces by implementing an autonomic computing approach to manage IoT environments. The goal is to minimize manual effort while ensuring a desired level of security through the development of an autonomic manager. This manager will continuously monitor the smart space, analyze contextual information, and respond to potential security threats, thereby reducing liability and risks of security breaches. To achieve this, the researchers adopt the microservice architecture pattern and introduce a comprehensive ontology named Secure Smart Space Ontology (SSSO) based on RDF triples. This ontology allows for the description of physical infrastructure, facilities, services and contextual information in security-enhanced smart spaces. Building upon SSSO, they design an autonomic security manager consisting of four layers. The manager continuously monitors the managed spaces, analyzes the situation, and takes appropriate actions based on the defined policies to maintain the desired security level. The paper presents Secure Smart Space Ontology (SSSO), which incorporates service-oriented, security-enhanced, event-driven, and context-rich features for IoT smart spaces. The proposed Autonomic Security Manager is developed as a microservice running in the smart space environment, following the MAPE-k method. The manager consists of four layers: Resource and Context, Triple Store, Manager, and Interface. The Resource and Context layer represents the physical infrastructure and contextual information, while the Triple Store layer stores RDF triples. The Manager layer implements the MAPE-k method, monitoring events, analyzing situations, planning actions, and executing commands. The Interface layer provides an API for interaction with the manager. The adaptive security policy allows for dynamic access control based on real-time data provided by services, written in SPARQL. The security manager queries the RDF graph to gather relevant information for analysis and decision-making. The evaluation includes a model of a smart conference room with 32 devices, 66

services, 30 potential threats, and 28 adaptive policies. The autonomic security manager effectively responds to a series of 160 events, maintaining security through the MAPE-k method. The manager responds to events within two seconds, demonstrating its suitability for large-scale smart spaces. In conclusion, the proposed SSSO and Autonomic Security Manager offer an effective and scalable solution for secure IoT smart spaces. The ontology enables comprehensive description and management of services, while the manager ensures adaptive security control through policy evaluation and event-driven decision-making. The evaluation validates the system's capabilities and efficiency, making it suitable for various smart space scenarios.

2.2.2 Discussion of Related Work

In previous research, concepts such as the MAPE-K model [70, 71, 79, 84, 90,91,93], ECA paradigm [73, 74] and high-level policies were very central because in order to manage a system autonomously, we need to know what the goals of the whole system are, then we need to set rules to determine what action should be taken if a condition is met. In addition, the management system needs to be able to monitor the computational environment, analyze its state, plan the actions to achieve a desired goal, and then execute the planned action. These actions typically require access to shared knowledge. As noted, these steps are modeled as multiple control loops in the MAPE-K model.

Most of the previous research on autonomous management has focused on specifically one or two aspects of the computational infrastructure of a smart city. Some of the research papers proposed an autonomic management system to organize the order of the operations that the administrator is going to perform on the devices, such as firmware update or diagnosis and the impact of those actions [77, 78]. In some cases, device management and how to make the devices discoverable, self-configurable and self-healed were also areas of attention [70, 71, 77, 78].

Gurgen [70] introduces a middleware that provides services for monitoring the city data and performs actions identified by various applications, but it lacks prototype development, experiments, and case studies. It briefly covers the middleware's functions, such as data collection and processing, composition of sensor and actuator services, and autonomic

management. However, it lacks detailed technical implementation. The study's classification as autonomic computing and smart city is somewhat misleading, as it primarily focuses on offering an autonomic life cycle management system for smart city applications rather than an automated management system for the smart city as a whole. Overall, the research would benefit from including practical examples, delving deeper into technical implementation, and addressing smart city management. In subsequent work, Gurgen et al. [73] mainly focused on the management of the sensor attributes and performance. Although some rules are defined for managing performance metrics, such as network parameters, it neglects to explain their adaptability over time in response to evolving system requirements. Additionally, while the paper addresses distributed hierarchical architecture and rule evaluation across various levels, it falls short in considering an analysis of the system's scalability concerning the growing number of sensors and managed elements.

The work by Braten et al. [71] was specifically able to manage two aspects of the smart city, devices and the whole system, which was interesting. This paper presents a general model for autonomous IoT device management. Additionally, it employs this model in a management system for solar-powered air quality sensing devices. However, the paper falls short in providing essential elaboration on both the specifics of the experimental environment and the intricate details involved in the development of the prototype. The primary emphasis is placed on device management, while infrastructure management is notably absent from the scope. Additionally, the study fails to address the crucial aspect of scalability, which is vital for assessing the model's applicability in larger, more complex IoT ecosystems.

Kyriazis [72] proposed a conceptual architecture that primarily focuses on the components within an IoT network, but does not include any prototype implementation, experimental evaluation, or considers infrastructure performance. They note that enhancements are needed to ensure practical implementation and optimal system efficiency.

The work by Sharrock et al. [74] focused primarily on sensing devices, while our objective revolves around the smart city ecosystem. We aim to consider a broader range of performance metrics beyond just sensor battery life, encompassing CPU and memory usage. Furthermore,

we plan to incorporate the smart city environment to ensure a comprehensive and effective solution.

Arzo et al. [76] investigated network automation and divided a complex network into atomic and autonomic units that could interact with each other to improve reusability and scalability. The work does not consider the smart city context and so the model's application for monitoring and managing the urban environment remains unexplored. Mezghani et al. [77] introduce a coordinator as a mediator for isolated DM (Device Management) platforms, facilitating seamless execution of device management operations across multiple device fleets. The main emphasis lies in task coordination in IoT networks, with no focus on the environment, or performance aspects. The proposed technique employs graph-based methods and includes monitoring, analysis, and planning features. Ayeb et al. [78] also focus on device management and coordinating device operations, primarily fault detection. Their approach allows administrators to monitor performance, but it lacks the capability for autonomic performance management.

The work by Kosińska et al. [79, 85] was on the autonomic management of cloud-native applications. Their work was the most related one to our objectives and it demonstrated the feasibility and advantages of autonomic management of complex environments. They set some high-level policies, and the framework could adapt and reconfigure runtime in real-time based on predefined management policies in the form of event-condition-action. This capability serves as a proactive measure to mitigate potential SLA violations. As an experiment, they outlined two rules for a policy governing the adjustment of CPU requests for Pods. They also present an experimental evaluation of their work. Having said that, their work concentrates on Cloud applications and does not explore the potential implications for smart cities and their environments. Consequently, there's a gap in addressing sensor measurements and attributes.

The work by Saadon et al. [80] recognizes the possibility of defining multiple rules simultaneously to support decision making for resource allocation but asserts that only one applicable rule will be chosen, though certain scenarios may require multiple rules. However, the study lacks a thorough exploration of potential interactions or conflicts between these

rules. Furthermore, the research discusses the function's adaptability but neglects to address how it will be updated over time to accommodate new rules and regulations, which is essential for maintaining its relevance and usefulness. Notably, the research overlooks smart cities in its approach.

Lam et al. [81] propose dynamic orchestration and configuration services in Industrial IoT Systems. It presents an approach for real-time adaptation capabilities. Nonetheless, a notable limitation is the lack of emphasis on performance metrics and operational management. Although memory consumption is monitored for comparison, the authors miss the opportunity to leverage this data for effective memory management. Additionally, the paper's focus on the manufacturing domain raises questions about the generalizability of the proposed approach to diverse industries or domains with varying requirements and characteristics such as smart cities.

The paper by Sampaio et al. [82] primarily centers on power consumption, but neglects to consider the unique aspects of the smart city environment and its performance of its infrastructure. There is a lack of explanation on how their approach could be effectively applied to large-scale IoT energy consumption scenarios. The research's narrow focus on a fire alarm system within a smart condominium restricts its exploration of potential applications or adaptability to other diverse IoT contexts.

Nalinaksh et al. [83] only discuss existing tools that support the autonomic IoT ecosystem but falls short in proposing concrete solutions for autonomic management and real-world deployment and in the context of smart city management.

In [84], de Sousa et al. utilized the MAPE-K model and policies for network management. However, a crucial aspect left unaddressed is the detailed explanation of how real-time network data will be collected, processed, and utilized for decision-making, monitoring, and policy enforcement. Also, the research does not discuss the issue of scalability, and there is a limited explanation of how the proposed solution can adequately handle the increasing network demands, growing traffic, nodes, and domains. Additionally, the study's focus solely

on network management limits its potential impact, as it does not explore the broader applicability of the approach to smart city environments and IoT networks.

In [86], Villela Zavala et al. acknowledge the significance of the self-configuration autonomic property within the IoT Autonomic Architecture, which is crucial for addressing scalability, application domain diversity, and device heterogeneity. However, the specific mechanisms by which this self-configuration property achieves scalability are not clearly explained. As the IoT device and data volume increase, the framework's ability to handle a large number of connected devices becomes critical. The study does not present experimental results to evaluate the operation of their system. Despite the title mentioning network management, there is a lack of elaboration on how the network itself is managed.

Riker et al. in [87] proposed AGREEN. One of the limitations of AGREEN's approach lies in its reliance on the fuzzy decision system, which may not be universally suitable for all DIoT environments due to variations in input data accuracy and the quality of fuzzy rules, potentially adding complexity. Thus, customizing the system for different DIoT environments might be necessary. Another potential drawback is its scalability for large-scale DIoT networks, as it necessitates a monitoring group of IoT devices sharing similar characteristics, which could prove challenging in diverse networks with numerous devices and applications. Furthermore, AGREEN focuses on the management of IoT device communication, encompassing communication intervals and active devices, yet it overlooks the smart city environment and infrastructure.

In research by Singh et al. [88] explicit detail about the experimental analysis is missing. While the researchers did create a pool of requests involving five autonomic machines across three distinct scenarios, they did not elaborate extensively on the specifics of these scenarios. This absence of detailed explanation leaves a gap in the understanding of the experimental design and its implications. Additionally, integrating the autonomic engine into existing cloud infrastructures and ensuring compatibility with diverse cloud service providers could present complex challenges that require meticulous planning and testing. Moreover, the study overlooks the considerations specific to smart cities and their environments. Furthermore, the evaluation process also lacks detailed elaboration.

The work by Shukla et al. [89] offers an overview of the challenges and high-level components of autonomic cloud resource management in Industry 4.0. However, for a more comprehensive evaluation, additional information is required, such as detailed insights into the algorithms, methodologies used in resource allocation, fault detection, and optimization, as well as concrete experimental results and real-world use cases. While a resource management diagram is provided, there is a notable absence of an explanation regarding resource allocation methods. Additionally, although the authors mention SLA violation and energy utilization, the desired or acceptable levels for these metrics remain unspecified. Moreover, the context of the research lacks a focus on smart cities and their environment.

The research presented by Mangla et al. [90] is notable for its implementation of MAPE-K for autonomic management. The focus ~~is~~ on resource management at the fog layer, rather than the cloud, represents a notable advantage of their approach. However, to fully evaluate the proposed architecture's feasibility and effectiveness, more detailed technical information and concrete examples are needed because their work lacks insights into the implementation of their approach and how they define the policies, and how the execute function governs the execution of actions. The mechanisms for resource allocation and optimization should be further elaborated upon to provide a clearer understanding of their functionality. While the authors suggest that incorporating the fog layer can address scalability, a more explicit rationale for this claim would be beneficial, especially when considering the accommodation of a large number of devices or complex IoT deployments. Additionally, the research lacks attention to the management of performance and operational aspects of the environment, which could be a valuable aspect to explore.

In [91], Patibandla et al. employed MAPE-K and utilized both public and private clouds, with a primary emphasis on resource management within the cloud environment. While they briefly mentioned the autonomic computing concept, they did not fully leverage it in the proposed algorithm; There is a lack of detailed insights into the formulation and definition of policies, which play a crucial role in guiding the system's decision-making process. Moreover, the research context does not explore smart cities and their ecosystem, limiting its scope and potential applicability in such dynamic urban environments.

SeaNet [92] is a methodology for autonomic network management in software-defined networks (SDNs), making use of graphs to facilitate the process. This research stands out due to its exceptional attention to scalability, highlighting it as a significant advantage of the approach. The evaluations include diverse topologies and host numbers, providing valuable insights into the system's performance. However, the study could benefit from exploring the broader applicability of SeaNet across various network technologies and industries. Currently, the generalizability of different contexts remains relatively unexplored, including the management of smart city networks and environment.

The research proposed by Lin et al. [93] primarily centers on security management, offering an automatic security manager based on MAPE K that effectively safeguards smart spaces. The authors also defined policies for security assessment, providing a well-explained approach and architecture. However, due to the unavailability of suitable testbeds for microservices in regular IoT systems, they were unable to evaluate the solution in such contexts. While their focus on security in smart spaces is interesting, the study does not consider other critical criteria relevant to smart spaces.

While previous research has addressed some aspects of autonomous management there are still significant gaps. There is a need to develop an autonomous management system capable of managing the smart city ecosystem - both its environment and infrastructure with a focus on performance and operational management. A smart city environment may collect data about environmental factors such as light, temperature and pollution, but the smart city infrastructure includes resources and metrics about the smart city architectural elements. As we will explain in detail in the next chapter, our system can monitor and manage the smart city environment as well as smart city infrastructure by checking the performance metrics and fixing the problems that happen in regard to those metrics. Our proposed policy-based system observes the key operational metrics and tries to improve and optimize them by performing managerial actions. Our system also can monitor the smart city environment factors and manage them by being in touch with the IoT platform and use of smart city applications.

Chapter 3 Smart City Model

As discussed in the previous Chapter, rapid urbanization across the globe has led to a plethora of challenges for cities, including population growth, massive number of smart devices, and social inequality. In response to these challenges, the concept of the "smart city" has emerged as a model for urban development that leverages technology and data to improve the quality of life of its citizens. Smart cities are characterized by the use of advanced technologies and data analytics to optimize infrastructure, enhance services, and engage citizens in decision-making processes.

In this Chapter, we will dive deeper into our smart city model, exploring its key features, benefits, and challenges. Firstly, we will introduce the general smart city model, which has become widely accepted among researchers. This model typically consists of several interrelated components, including the smart city entities, an IoT platform and smart city administrators/managers. However, we believe that this model should be extended to include additional elements that are essential for a truly smart city, and we present our extended version of the model.

To truly embody the concept of a smart city, it is important to prioritize not only the optimization of infrastructure performance but also ensure the monitoring and autonomous management of the city's systems. Infrastructure performance is essential for ensuring the smooth functioning of a city's critical systems. However, merely optimizing infrastructure performance is not enough to ensure that the city can efficiently handle unexpected events or adapt to changing circumstances. Constant monitoring of a city's systems allows for the early detection of issues before they become major problems. With the help of advanced sensors and monitoring technologies, it is possible to monitor a wide range of factors such as the status of the smart city resources, services, and processes. By analyzing this data in real-time, city officials can quickly identify potential issues and take proactive steps to mitigate them before they escalate. In addition to constant monitoring, autonomous management can significantly improve the operation of a smart city. A smart city operates around the clock, making it essential for management to be available 24/7. Automating management processes can help reduce human intervention and so can help the city function more efficiently and cost-

effectively. Furthermore, autonomous management can also help to reduce human error, which is a common cause of system failures, and also free up human resources to focus on more complex tasks. However, it is important to ensure that the autonomous systems are properly designed and tested to prevent unintended consequences or safety risks.

We first review the terminology we use in this research:

- A ***smart city environment*** is based on geographical and physical viewpoints and frequently involves monitoring of environmental factors such as humidity, pollution, temperature, and light that the sensors can measure.
- ***Smart elements, objects or things*** are general terms that include any sensors or devices that exist in a network and are connected to each other via a communication protocol.
- In the realm of smart cities, sensors and devices play pivotal roles in creating interconnected and intelligent urban environments. ***Sensors*** serve as specialized components capable of measuring various physical factors, including temperature, humidity, motion, light, and more. These sensors act as data collectors, providing valuable information about the surrounding environment and converting it into electrical signals or readable outputs. On the other hand, ***devices*** in a smart city encompass a wide range of machines, including smartphones, smart appliances, gadgets, security cameras, and more that have multiple functionalities and can interact with users or other devices and perform specific tasks. They can be typically equipped with various sensors, processors, applications, and interfaces to execute their intended operations.
- The ***smart city infrastructure*** refers to the technology architecture, components, sensors and devices, and connections that make up a smart city system. Components are the fundamental building blocks that collectively form the intricate framework of a smart city architecture such as computational devices, network equipment, and data centers. In essence, the smart city architecture encompasses an amalgamation of diverse components interconnected to enable seamless functionality of the smart city. These components play a pivotal role in shaping the technological and infrastructural landscape, empowering the realization of a truly intelligent urban environment. These

components can be computing nodes, energy management systems, storage units, and data centers.

- The ***smart city ecosystem*** encompasses all elements of a smart city, from the smart city environment to smart city infrastructure. This term is used to describe all the aspects of a smart city such as infrastructure performance metrics, sensor attributes and sensor measurements about the city environment (temperature, air quality, and more)
- ***Telemetry data*** refers to the data that is generated by the sensors and devices in a smart city and can include environmental data, traffic and transportation data, energy consumption data, public safety data and more. Whereas the ***operational data*** is the data about the smart city infrastructure and is related to its daily operations. This data provides insights into the current state of different components such as IoT platform, storage, data filtering unit etc. The analysis of this data yields invaluable insights into the smart city infrastructure and the seamless operation of its interconnected systems.

3.1 Smart City Features

Smart city features refer to the advanced technologies and innovative solutions that are integrated into the urban environment to differentiate smart cities from traditional urban environments. These features include but not limited to:

- ***Digital Infrastructure***: Smart cities rely on advanced digital infrastructure, including high-speed internet connectivity, cheap and ubiquitous sensors, and data analytics tools, to provide high quality services for their citizens.
- ***Online services***: In a smart city, companies, organizations, institutions, and even the government provide their services through online platforms.
- ***Data Analytics***: Smart cities use data analytics to monitor and analyze urban systems and services, enabling real-time decision-making and optimization.
- ***Integrated Services***: Smart cities aim to integrate various urban services, such as transportation, energy, health, and waste management, to improve efficiency and reduce environmental impact.

- ***Citizen Engagement***: Smart cities encourage citizen participation in decision-making processes and service delivery through digital platforms and community engagement programs.

3.2 Smart City Benefits

Smart cities have the potential to revolutionize the way we live and work. Smart cities offer a range of benefits for their residents. Some of the key benefits of smart cities include:

- ***Improved Efficiency***: By using the data that comes from a smart city and analyzing it, the smart city services can be more optimized and cost efficient.
- ***Enhanced Quality of Life***: In smart cities citizens can enjoy better quality of life due to better access to more efficient services and facilities such as health, education and transportation.
- ***Increased safety***: Because in a smart city some technologies such as surveillance cameras, smart traffic management systems, and emergency response systems are used, public safety is enhanced.
- ***Environmental Sustainability***: Smart cities benefit from cutting-edge technologies to provide more efficient waste management, decrease carbon emissions and reduce energy consumption therefore they can help to reduce environmental impact and promote sustainable development.
- ***Economic Development***: Smart cities can lead to significant cost savings for cities and a more efficient use of resources therefore smart cities can drive economic development by creating a favorable environment for businesses and encourage foreign investment.

3.3 Smart City Challenges

Although smart cities can improve efficiency and safety, quality of life of the citizens, and also provide environment sustainability and develop the economy, due to their complexity and multifaceted nature they face a range of challenges that need to be effectively tackled through appropriate solutions. While acknowledging the breadth of challenges associated with smart

cities, it is important to recognize that our research may not comprehensively address all the following challenges. Some of the key challenges include:

- Integration: Smart cities face the challenge of ensuring seamless communication and cooperation between numerous organizations, institutions, and public and private sectors, all of which may have different standards and technologies. This requires establishing compatibility and interconnectedness between various systems, allowing for efficient sharing of information and resources. Failure to address this challenge can hinder the effectiveness of smart city initiatives and create barriers to innovation and progress.
- Data Privacy and Security: Another challenge that smart cities should deal with is data privacy and security. In smart cities, devices and sensors and other components are connected to each other and send and receive data using communication protocols and this can raise concerns around privacy and security. Not only do we need to make sure that the communication channels are secure, but we must ensure that all of the components and devices are protected against attackers and security breaches. Other than this in a smart city we should make sure that the users are authorized, and the right users have access to the right data. However, this is one of the challenges that falls outside the scope of our current research.
- Infrastructure Investment: Although we mentioned that smart cities can save us money and are cost efficient, the development of the advanced digital infrastructure needed for smart cities at the outset can be costly and require significant investment.
- Equity: Smart city initiatives must ensure that all residents have equal access to smart city services and benefits, regardless of income or social status.
- Citizen Engagement: All of the smart city citizens should be able to engage in the decision-making processes equally to ensure that their needs and perspectives are taken into account.

3.4 A Smart City Model

In this section, we introduce a high-level model of a smart city that forms the foundation for our research. We first introduce a basic model of a smart city that has emerged from much of the related research around smart cities. We then present an extended version of this basic model that we have adopted for our work.

3.4.1 A Basic Smart City Model

A basic smart city model is illustrated in Fig. 3.1. It is designed to keep the urban environment running smoothly. This model is composed of multiple layers, including devices and entities as well as an IoT (Internet of Things) platform or device management platform. This basic model serves as the cornerstone for numerous research papers on smart cities [53, 54, 57, 58, 59]. Research into the development of smart cities has moved toward ensuring that a critical component of a smart city is some platform, here called the IoT Platform, that is specifically devoted to helping track, monitor and manage the sensors and end point devices in a smart city. The rationale for the platform is the fact that smart cities are expected to make use of hundreds or thousands of sensing devices and that tools are needed to help deploy and manage those. As discussed in Chapter 2, a number of IoT Platforms have been proposed having similar but also different characteristics, though the overarching goals are the same. The third layer in the model consists of the smart city manager (team) who oversees everything and is responsible for ensuring that the smart city environment and its various smart elements are functioning properly. This includes monitoring environmental factors like temperature and light, as well as the status of sensors and devices, telemetry data, and even alarms and errors. By monitoring everything closely, the manager can detect potential problems quickly and resolve them before they become more serious, ensuring a stable and reliable smart city.

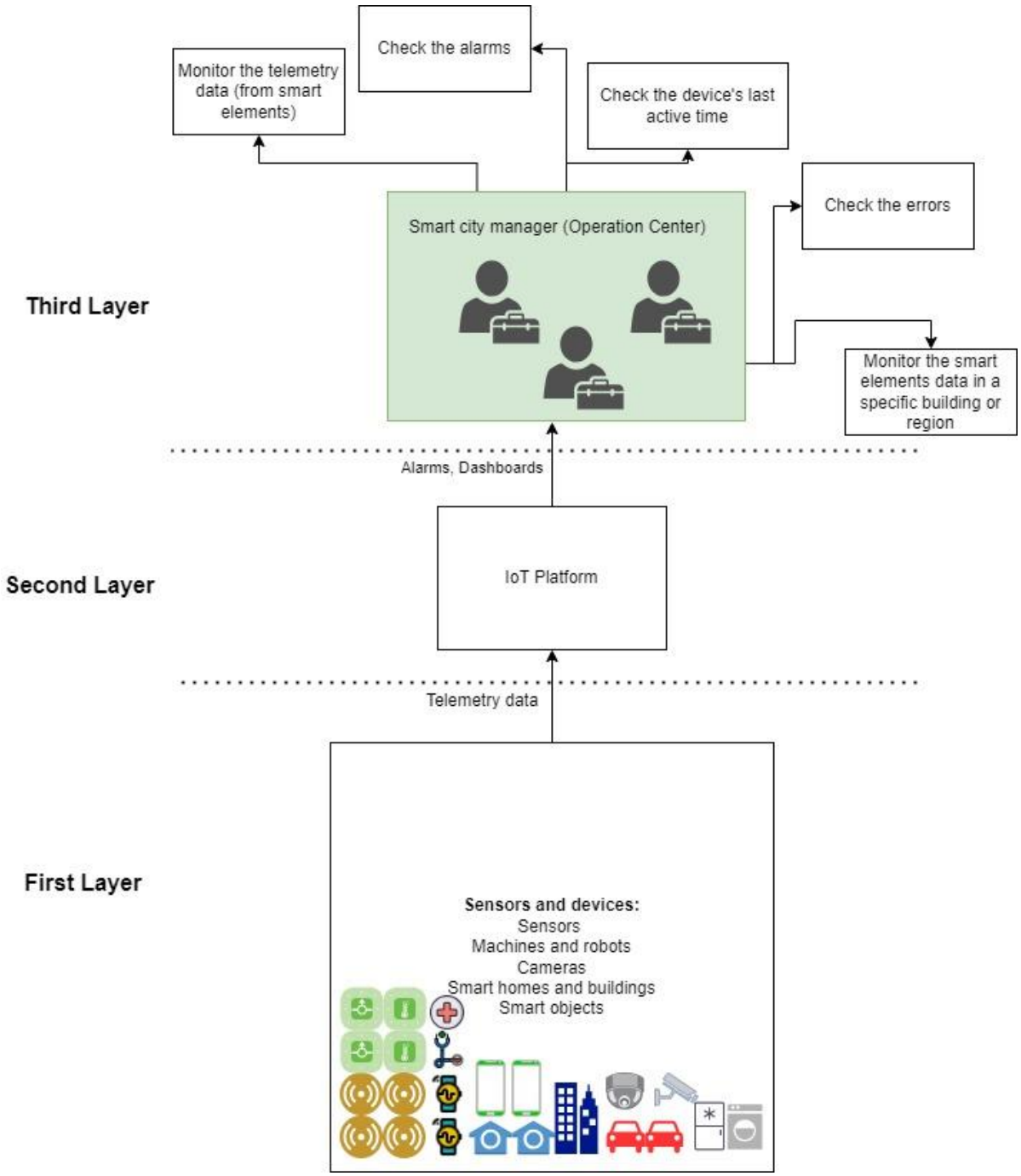


Figure 3.1 Basic Smart City Model.

The basic smart city model serves as the foundation for a more advanced and sophisticated model that we will discuss later. The first layer of the basic smart city model (see Fig. 3.1) includes sensors and smart devices or things such as machines and robots, cameras, etc. that are deployed throughout the city. In a smart city, sensors and devices can be integrated to

support a variety of applications and services, such as smart health, smart transportation, smart energy, smart agriculture, and more. We have also included other devices, such as cameras and robots, as part of this first layer. Once the sensors and devices collect data, the data is then directed to the next component in the smart city model. To enable this transfer of data, there needs to be a robust connectivity infrastructure in place. This connectivity is based on communication methods such as MQTT, CoAP, RFID, 5G, or other communication protocols. These sensors and devices generate streaming and fluctuating data. This data is collected, processed, and visualized by the IoT platform or device management system. The device management platform should have the following capabilities: it should aggregate the data from a variety of different devices, process it in real-time and in different volumes, support heterogeneity, provide data visualization, generate alarms and notifications, assign devices to specific assets, and manage user's assets, devices, and dashboards. The city's management team is responsible for deploying applications and assessing the status of the city, monitoring the data from sensors and devices, perhaps even monitoring aspects of specific city buildings. Also, they must regularly check the operational status of the city's infrastructure, such as the attributes of the devices, such as when they were last active or connected to the network, network traffic, etc. and must monitor alarms and errors generated by the system and take appropriate actions.

Through the utilization of this basic model, we created a prototype for simulating a smart city. Using this smart city prototype, we conducted a series of experiments, effectively managing diverse facets of the smart city, including its intricate array of sensors and devices and also generating alarms based on data fluctuation. It is worth emphasizing that this fundamental model played a pivotal role in our research endeavors, culminating in the publication of a paper that showcased our findings and advancements in this domain [94].

The model depicted in Fig. 3.1 is designed to primarily provide data from the devices in a smart city. However, it is also necessary to monitor the different components of the smart city, including the connections between them, the computing nodes, and the fog and cloud infrastructure. We must keep track of resources, services, processes, and other important elements of the smart city's infrastructure. By monitoring all the different components of the

smart city, we can detect potential problems and take appropriate actions to prevent issues from escalating. For example, we can identify and resolve issues related to network connectivity, server performance, and software glitches. This can help to ensure that the smart city operates smoothly and provides a high-quality living environment for its residents.

3.4.2 An Extended Model of a Smart City

Our focus is on the infrastructure and operational components of the smart city. While an IoT platform can help with some aspects of managing sensors and devices in a smart city, we also want to be able to manage all of the services and processes that make a smart city function effectively. Our ultimate goal is to create an autonomous and sustainable monitoring and management system that can support a smart city where citizens can thrive.

In order to achieve a more comprehensive view of the smart city's infrastructure, we have expanded the smart city model beyond the IoT platform and added a monitoring and instrumentation platform. This platform is designed to provide a more detailed level of monitoring of the broader infrastructure and its components. The monitoring and instrumentation platform can extract performance metrics from various components, applications, and services, including getting data from the IoT platform. This framework should be capable of collecting data on various performance parameters, such as response times, throughput, and CPU usage. It enables the monitoring of different components, such as servers, network devices, and software applications, which helps to provide a more comprehensive view of the smart city's operational infrastructure. Moreover, the expanded smart city model offers a centralized view of the infrastructure, which enables administrators to identify performance trends and patterns. By monitoring aspects of the smart city infrastructure, we can detect issues and identify potential problems before they escalate into critical situations.

The smart city model used in our research is shown in Fig. 3.2. This extended model provides a more comprehensive view of the smart city's infrastructure, including dependencies, internal components, and connections between various components. As shown in the figure, the extended model provides more detailed information on the operation of the smart city

infrastructure, which enables the management team to understand how different components work together.

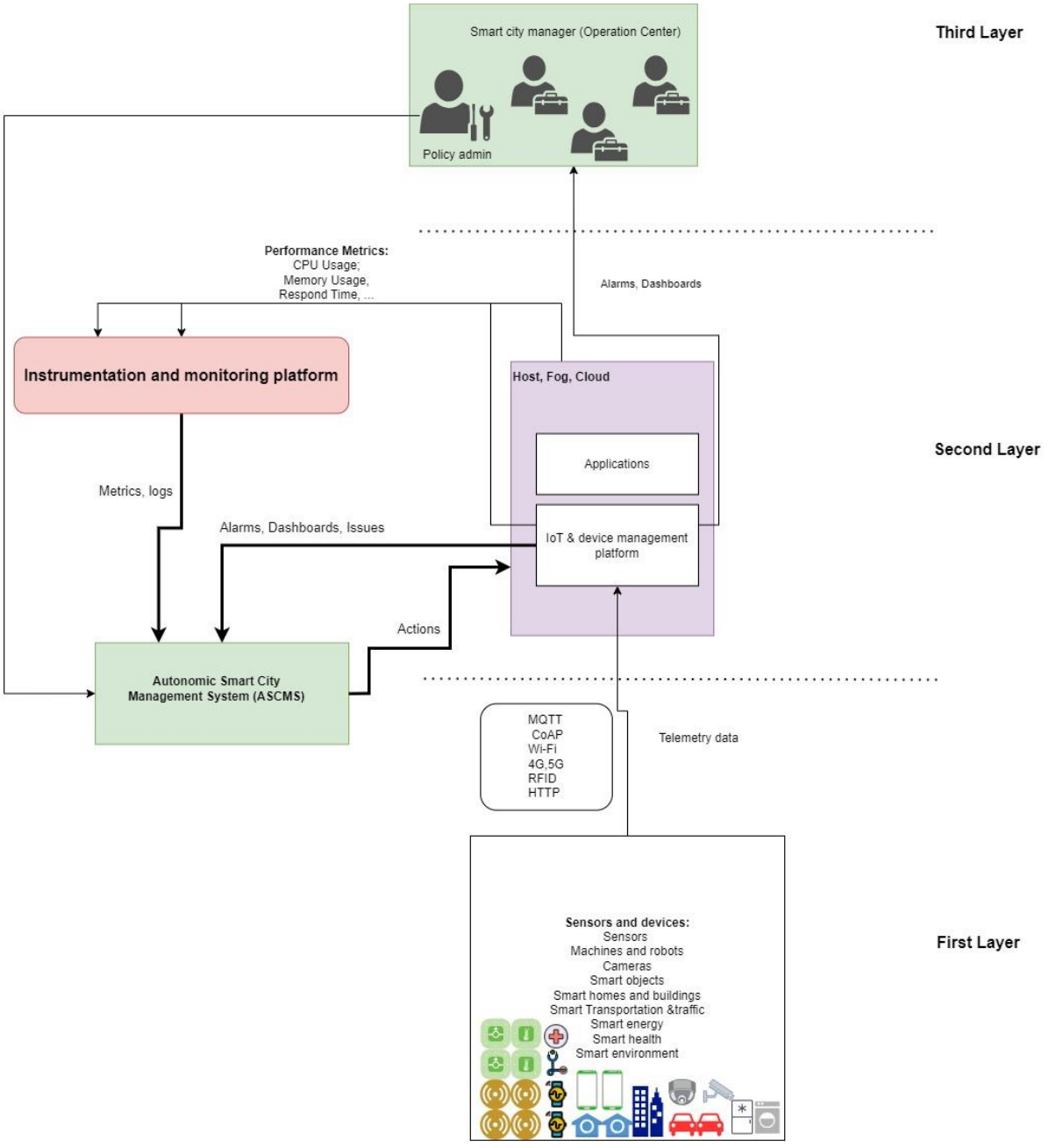


Figure 3.2 Refined Smart City Model.

By understanding the dependencies between components, we can identify potential bottlenecks or points of failure and take preventive measures to ensure that the smart city infrastructure runs smoothly. In addition to the dependencies and connections, the extended

model should also include information about the internal components of the smart city infrastructure. This information is crucial in understanding the functioning of the different components and their impact on the overall performance of the smart city. The extended model also assumes the capability to monitor applications. One of the types of applications that we assume to be present in the smart city infrastructure is a data filtering application which is used to filter and classify the telemetry data and delete unwanted data before reaching the IoT platform. This unit ensures data cleanliness before entering and populating the database, optimizing resource usage, a critical aspect in the resource-constrained environment of a smart city, where vast amounts of data are handled.

In this model, the first and second layers are taken as given in the basic model. We have added a monitoring and instrumentation platform to the basic model in order to monitor the components of the infrastructure and send the operational data, performance metrics and logs to the smart city management platform. These multiple streams and variety of data from devices, applications, infrastructure, etc. create enormous problems for humans involved in monitoring and management of the smart city infrastructure. While additional staff could be added to help manage the smart city infrastructure, at additional ongoing cost to the city, the complexity of the infrastructure is not so easily addressed. Hence, some form of automation will be needed.

In order to help address managing the operational aspects of a smart city, we proposed adding a monitoring and instrumentation component to create a new model with autonomic smart city management system, as illustrated in Fig.3.2. The monitoring and instrumentation platform could then send the operational data, performance metrics, etc. to the autonomic smart city management system to be processed and analyzed based on some predefined policies. Our Autonomic Smart City Management System (ASCMS) architecture takes inspiration from the MAPE-K model, which we will delve deeper into in Chapter 4. The MAPE-K model consists of four core components: Monitor, Analyze, Plan, and Execute. Each of these components has access to a shared knowledge base that helps them make informed decisions. For the monitoring component we use the monitoring and instrumentation platform to extract the performance data from the infrastructure along with some information from the IoT platform.

The autonomic smart city management system has several important duties to ensure that the smart city runs smoothly. These include checking the status of the smart city's infrastructure in real-time, making sure that all the sensors and devices are up-to-date with the latest software updates and security patches, fixing any connectivity issues that may arise, keeping track of performance metrics like response times and bandwidth, optimizing resource usage like CPU and storage space, resolving any hardware and software problems that occur, and providing centralized management for sensors and devices at a high level. These management tasks are done through the policies that drive the autonomic management system. By executing these tasks, the autonomic smart city management system can help ensure that the smart city is functioning efficiently and effectively.

What makes a smart city different than the other networks and its management more complex than others is the scale of it. A smart city includes many smaller networks, distributed systems, massive number of devices and sensors, applications, cloud platforms and so on. Therefore, its management is more complicated compared to other networks and systems. This management has many aspects such as sensor data management, performance management, resource management, failure management, etc. Developing an autonomic management platform encompassing all these aspects is a long-term initiative. In this research, we begin with an initial focus on autonomous performance management of the operational side of the smart city, aiming to establish the foundation for further research.

In Chapter 5, we will present a smart city prototype that is based on the refined smart city model. This prototype will serve as our smart city simulator, and we will conduct experiments using this prototype to shed light on the management requirements of such a system. Specifically, we will monitor and visualize the telemetry data from simulated smart city devices in the prototype, providing insights into the performance of the system and identifying areas for improvement.

Chapter 4 An Autonomic Management Model for Smart Cities

The Autonomic Management Model is a framework that aims to manage and optimize the smart city environment and infrastructure autonomously. The model is based on the principles of self-organization, self-management, and self-optimization. It provides a systematic approach to control and manage the environmental aspects of a smart city, the complex interactions between the different components of a smart city, and the infrastructure components and performance as well.

Fig. 4.1 presents our model of the smart city along with the architectural framework of the Autonomic Smart City Management System (ASCMS), showcasing the interconnectedness and interdependencies between various components. Notably, the data filtering unit and IoT platform are seamlessly linked to the monitoring and instrumentation tool, allowing this tool to gather and analyze performance data from both. The monitoring and instrumentation tool serves a dual function: firstly, measuring operational metrics concerning these components and their internal elements, and secondly, promptly logging and reporting any infrastructure failures encountered. ASCMS receives these crucial metrics, failure data, and telemetry information, empowering it to effectively manage the smart city's infrastructure and environment. This management process is performed by applying specific rules and policies defined by the Policy Administrator. The smart city management system is designed with an essential feedback loop that plays a vital role in controlling and optimizing the overall smart city infrastructure, fostering a seamlessly operating ecosystem.

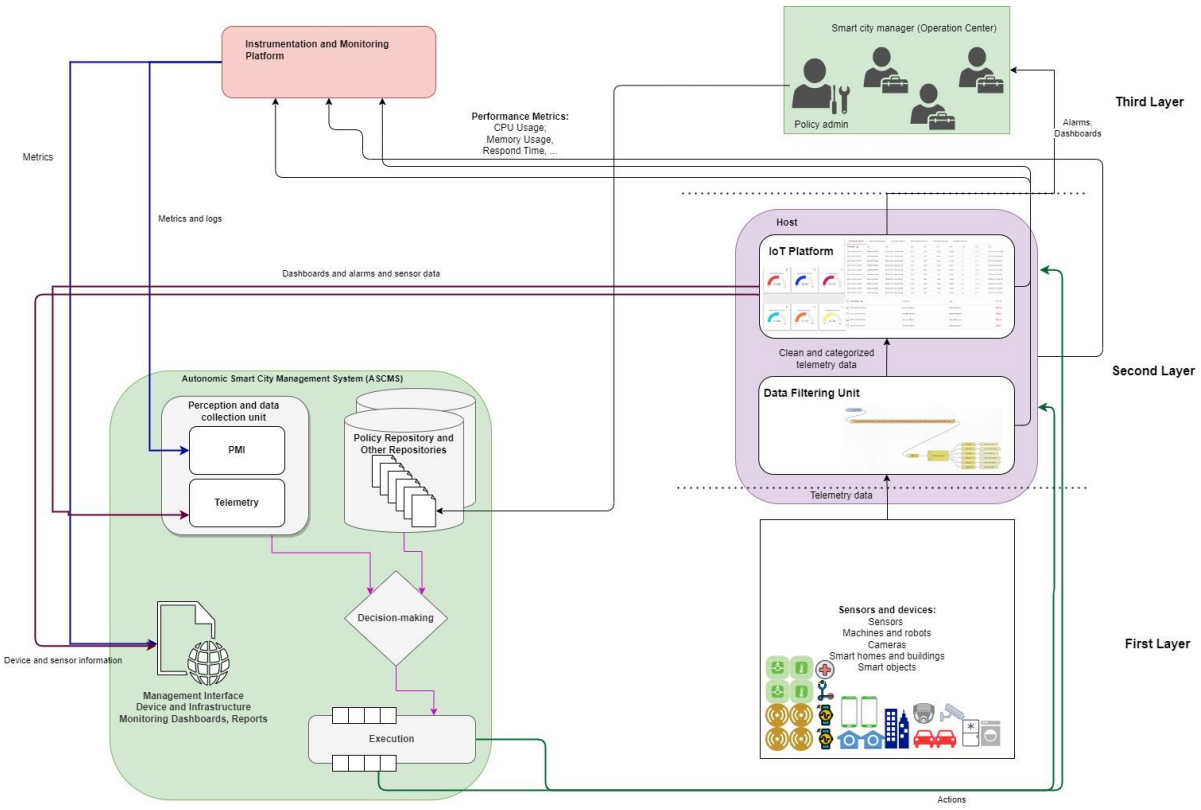


Figure 4.1 Autonomic Smart City Management System (ASCMS)

As mentioned in the previous Chapter our autonomic smart city management system architecture (as shown in Fig. 4.2) has a built-in loop which is inspired by the MAPE-K architecture to control the whole smart city ecosystem. In the following we provide a detailed breakdown that correlates each component of our model with its corresponding element in the MAPE-K framework:

Monitoring: The monitoring component monitors the environment and the smart city infrastructure. It furnishes essential performance metrics, resource utilization data, and comprehensive insights into the status of services and processes. In our research, we developed the "Perception and Data Collection Unit" to serve this objective. It takes two streams of inputs: the performance metrics from the “Monitoring and Instrumentation component” and the telemetry data and sensor attributes from the IoT platform.

Analysis: The analysis component is responsible for analyzing and processing the data collected by the monitoring component to identify patterns, trends and other information for

decision making process. The goal of this component is to gain some insights into the system's behavior and identifying the areas for corrective actions.

Planning: The planning component is in charge of figuring out what actions should be taken when the system is not working as it should be. High-level goals are used in this component that outlines what needs to be done to get the system back on track. In our Autonomic Smart City Management System (ASCMS) these policies are structured as Event Condition Action (ECA) rules. Each ECA rule describes what action should be taken based on a specific event and a particular condition. We developed the "Decision-making unit" with tasks comparable to those carried out by the analysis and planning components within the MAPE-K architecture.

Execution: The execution component is responsible for putting the plan into action. Once the planning component has identified what needs to be done, the execution component carries out those actions on the parts of the system that need attention. Notably, in our model, we harness the same component to assume the responsibility for executing actions within the smart city ecosystem.

Knowledge: The knowledge component is usually connected to other components and serves as a repository for information that can be used by the other components of the system. Within our ASCMS model, we established a policy repository tasked with housing policies and comprehensive information pivotal to the decision-making process; other information, such as firmware versions, are stored in other repositories used by the autonomic management system.

In our proposed management system, we extract the performance metrics from the smart city infrastructure using the monitoring and instrumentation component and manage the whole infrastructure from the top to the bottom layers using the stored rules and policies as shown in Fig. 4.2. After receiving the performance metrics and sensor data, the decision-making component analyzes them based on the conditions stored in the policy engine. The policy engine contains the rules and conditions for maintaining the desired smart city ecosystem. Each policy is stored as the following:

On (event) If (Condition) then (Action)

Based on the policies, an execution plan will be chosen by the decision-making component and performed under the supervision of the Execution component to keep the ecosystem balanced and try to optimize performance. For instance, in order to monitor and manage a delay in the system, we set a policy on the response time to control that metric or lower it.

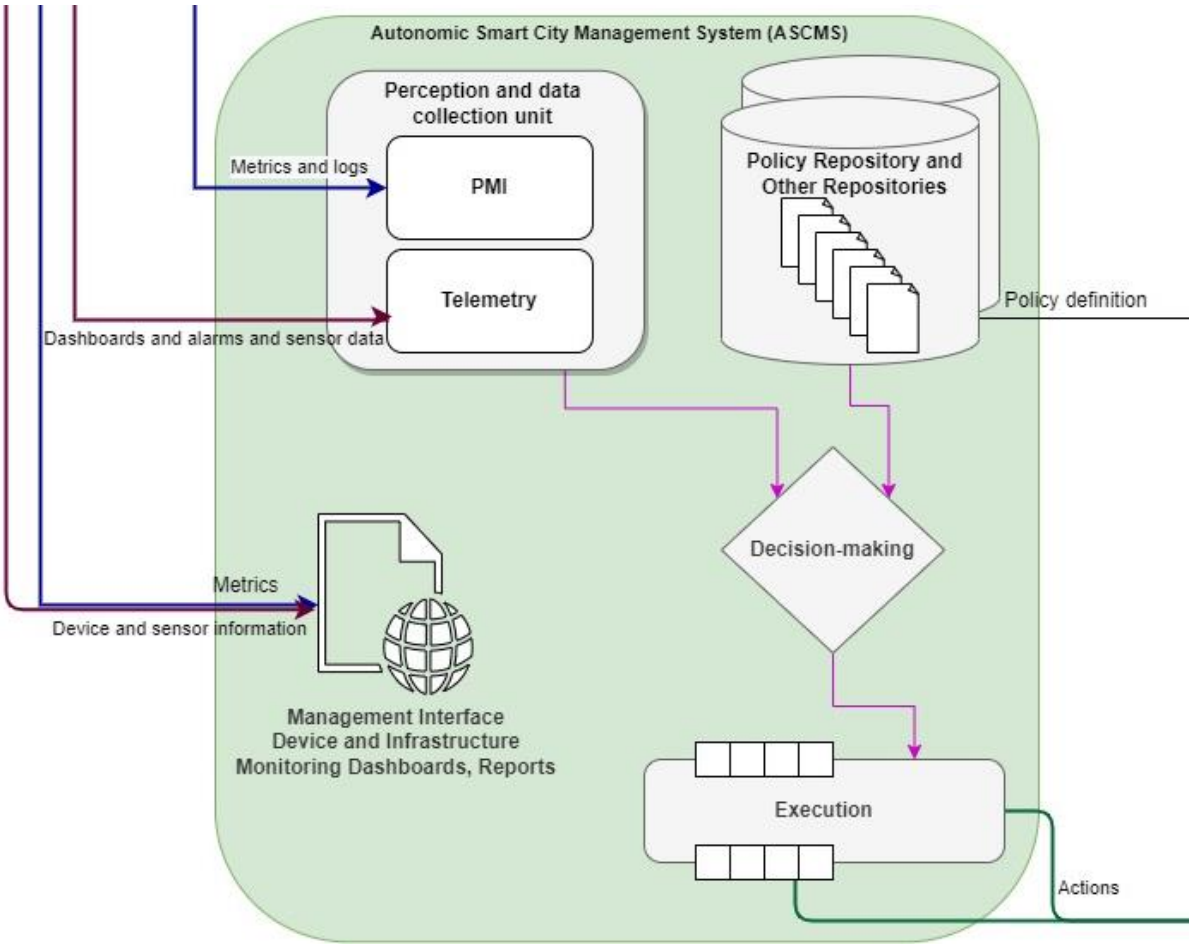


Figure 4.2 Autonomic Smart City Management System Model (ASCMS)

Detailed descriptions of each component are provided below, offering comprehensive insights into their functionalities and characteristics.

Monitoring and instrumentation component: This component plays a crucial role in continuously monitoring the smart city infrastructure and analyzing the performance metrics of various components and services in real-time. By leveraging advanced monitoring, this component provides valuable insights into the health and efficiency of smart city applications, infrastructure, and networks. This component measures and tracks essential factors that impact

the smart city's infrastructure performance and then transmits this data to the “Perception and data collection unit”. It can collect response times, resource utilization, error rates, and throughput, among other key metrics. These measurements offer a comprehensive view of the smart city's functionality. Real-time monitoring is a core feature of this component, allowing for immediate detection of anomalies or issues.

Perception and data collection unit: This unit is responsible for gathering and receiving data from various sources in the smart city. The perception and data collection unit is divided into two sub-units - Telemetry and Performance Monitoring and Instrumentation (PMI) - each with unique responsibilities. The Telemetry sub-unit is responsible for collecting data from various sensors and devices. This data contains information about the city's environment in real-time, such as the quality of air, traffic flow, and temperature. The majority of the telemetry data from the sensors might not be essential for management purposes, so it is directed to the IoT platform. However, there could be specific telemetry data that holds value for management. Hence, we selectively receive and integrate this significant data into our ASCMS. On the other hand, the PMI sub-unit receives the performance data about the smart city infrastructure. This sub-unit collects data about the performance of various components within the smart city infrastructure from the “Monitoring and instrumentation component”, including servers, storage devices, and processing units etc. This information is used to monitor the overall health of the smart city infrastructure and ensure that it is functioning efficiently.

Policy repository, management policies and other information: In order to manage a smart city autonomously, the first step is to define policies that govern the system's behavior. These policies can be created using policy language specifically designed for autonomic management systems. By utilizing this language, administrators can establish policies in event-condition-action format related to infrastructure performance, sensor measurements, and other relevant areas. Once these policies have been established, the autonomic management system can leverage them to make informed decisions about how to effectively manage the smart city. When a condition is evaluated as true, it signifies the occurrence of an event that triggers the activation of the corresponding policy. Subsequently, the system initiates corrective actions in accordance with the predefined policy, to ensure policy

enforcement. These policies play a critical role in ensuring the efficient and effective operation of the smart city and its infrastructure. They define the necessary actions that should be taken in response to various events, such as traffic congestion, power outages, or extreme weather conditions. By proactively defining policies, the smart city can quickly adapt to changes and ensure the safety and comfort of its residents. For instance, a policy might increase the frequency of public transport services during rush hour to alleviate traffic congestion, or automatically switch to backup generators during a power outage. These policies can be defined with thresholds, such as the maximum acceptable level of traffic congestion or the minimum acceptable temperature for heating systems. As a result, policies can be triggered automatically when these thresholds are exceeded, without requiring human intervention.

Repositories act as a central component within the system and serve as a centralized location for storing all information about policies and rules and other relevant information for decision-making and actions of the autonomic management system. Policies are comprised of sets of rules and guidelines that specify how the smart city system should behave under certain conditions. Each policy is stored in an event-condition-action format, which is a common practice for policy-based systems. The event part of a policy refers to a triggering occurrence that prompts the system to respond. The condition part defines specific criteria that must be met before the system takes action, while the action component outlines the precise response the system should undertake in response to the event and condition.

In addition to the policy repository, there may be other repositories containing information used by the management system. Some of this information could be used in decision making and other information might be necessary for management actions. The information would depend on the policies and actions defined for the management system.

Decision-making unit: The decision-making unit is connected to the perception unit and the policy repository and can process the data coming from the perception unit and makes decisions based on predefined policies and rules stored in the policy repository. This unit is a combination of the analysis and planning components in the MAPE-K control loop. The Decision-making unit’s inputs are the telemetry data, metrics, logs, alarms and high-level policies. It makes decisions based on the following condition:

If (adaptation is needed based on the policy) then (trigger execution unit)

One important aspect of the decision-making unit is the implicit loop that is used to ensure that the overall system is working as expected and the policies are met. This loop continually checks the status of the smart city factors and metrics and compares it to the desired state after the action has been performed. The ultimate goal of the decision-making unit is to create an autonomous smart city management system that can manage the city's environment and infrastructure without the need for human intervention. This goal can only be achieved through the constant monitoring and adjustment of the system, which is facilitated by the cyclical and recursive nature of the decision-making process.

Execution unit: The execution unit is responsible for carrying out the actions determined by the decision-making unit. This layer is the final step in the autonomous decision-making process. The execution layer is connected to the decision-making unit and receives the decisions made by the system. These decisions are then translated into actions that are executed in the smart city environment and infrastructure. To ensure the successful execution of actions determined by the decision-making unit, the execution layer must be equipped with essential resources and maintain control over the relevant infrastructure. This prerequisite necessitates a high degree of coordination and communication among all system components. Every time an action is executed, it requires the active involvement of all system components. The control loop should consistently iterate to ensure that the smart city ecosystem is steadily progressing toward its overarching high-level goals. This iterative process involves the synchronized effort of all components, ensuring the system's alignment with its strategic objectives.

Device and infrastructure monitoring web platform: This component is not strictly part of the autonomic management system per se. rather it provides an interface into the management system and is important for practical reasons and so is included in our model. The platform provides access to and displays valuable information and metrics about the smart city ecosystem in one place. This platform allows city administrators to monitor various devices, sensors and the whole infrastructure. It presents real-time data and metrics that help administrators gain insights into the overall health of the infrastructure and ensure that

everything in the smart city is under control. Using this web platform, city administrators can view and analyze different types of data in the form of charts, graphs, and tables, which makes it easy for them to interpret and draw conclusions. It can also display alerts and notifications in case of any critical incidents, allowing administrators to know what is going on in the smart city.

One of the key advantages of the autonomic management system is its ability to adapt to changing circumstances. The administrator can define multiple policies for one scenario and upon occurrence of each specific event the corresponding policy can be in effect. For example, for the temperature scenario the administrator can determine several ranges, and for each range, specific action can be defined. For instance, if the temperature is between t_1 and t_2 ($t_1 <$ temperature $< t_2$), the decision-making unit can determine to turn the cooling system on to decrease the temperature, and the execution unit can implement the changes immediately. Alternatively, if the temperature is too high (temperature $> t_2$), the decision-making unit could turn on the cooling system or take other actions (assuming that they are included in the policy specification) such as turning on misting systems or providing additional shade in outdoor areas to cool down the environment.

The autonomic management system also promotes sustainability by optimizing the infrastructure performance and the use of resources such as CPU and memory. It can reduce CPU consumption by optimizing resource allocation, load balancing, process scheduling, and performance monitoring. By implementing these techniques, the system can improve system efficiency and reduce energy consumption.

In the autonomic management system, the policies are defined and stored in the database beforehand. While monitoring the smart city components and infrastructure are being done using the monitoring platform, the management system selectively retrieves the metrics for which policies have been defined, facilitating a thorough analysis based on the specified policies. In this phase, the extracted metrics are analyzed and if a metric is based on a condition the planned action will be executed in the smart city infrastructure to adapt to the change.

Our proposed model provides smart city configuration and management through a rule-based engine that can automatically configure the smart city ecosystem and its infrastructure based on predefined policies. This model uses the control loop that can analyze the current state of the system and generate configuration actions accordingly that will be explained in detail in chapter 5. Similarly, our model provides healing capabilities by detecting and diagnosing issues within the ecosystem and infrastructure, and then implementing policy-based actions to resolve them. These actions may include restarting a service, configuring a sensor, assigning more resources, or modifying the smart city element's settings. Finally, our model can help with optimizing resource utilization and maximizing smart city operational performance. Our system continuously analyzes the smart city's behavior and measurements and performs actions to improve its performance. To demonstrate the efficacy of our model, we will create prototypes to simulate the smart city architecture and autonomic management system and conduct experiments that show how it can automatically configure, manage, heal, and optimize the smart city ecosystem and its infrastructure.

In the next Chapter, we present our smart city prototype and autonomic management system prototype. The smart city prototype includes the simulation of various sensors and devices, a data filtering unit and the IoT platform. We illustrate our smart city prototype with some simple examples. We then describe the prototype for our automatic management system, which will have an instrumentation and monitoring platform along with other important components. We also present a number of our management policies that will be stored in a database to help with the management of the smart city.

Chapter 5 Prototypes and Management Policies

In this Chapter, we describe our simulation of a smart city and our prototype autonomic management system. Since we do not have access to an instrumented city, we must rely on a simulation of a smart city and demonstrate management of the simulated environment by our prototype management system.

5.1 Smart City Prototype

In this section, we provide an assessment of our smart city model by constructing a) a simulation of a smart city and b) evaluating components that would form part of our management system. Since we do not have a smart city that we can use, we must rely on simulations to evaluate our overall approach. We first present our smart city prototype and some initial experiments that were used to gain some insights into how to do monitoring and a better understanding of some of the issues in management. This also provides background on the smart city prototype that was used for more extensive experiments with our management system.

5.1.1 Smart City Prototype

In our prototype, we explore some initial questions by creating a simulator that produces flows of data from sensors and devices in a smart city. We use this prototype to help us understand the rules and use of the IoT platform and to assess the use of a platform that we would be using for our experimental evaluation. The smart city prototype was composed of several different tools and components, as depicted in Fig. 5.1. It is based on the smart city model presented in Chapter 3 and encompasses two layers of that model.

The first layer consisted of devices and sensors, as described in our smart city model. The second layer includes the IoT platform and the processing components, such as a data filtering unit. In our prototype, the data from sensors and devices (telemetry data) is directed to the data categorization and filtering unit which is Node-Red [95], a flow-based programming tool. The data categorization and filtering unit is used to clean and send the data to our IoT platform (ThingsBoard) via the HTTP protocol; as noted in our smart city model we have assumed the presence of some data filtering components. In our prototype we also make use of Node-Red

to simulate various streams of data. To monitor the sensors, we have chosen ThingsBoard to be our IoT platform. ThingsBoard was chosen for several reasons:

- Supports popular communication protocols like MQTT, CoAP, HTTP, and so on.
- Extensive documentation, user-friendliness and robustness
- Customizable and extensible with the ability to add custom widgets.
- Device-agnostic, no need for special adaptation or interoperability.
- Capable of extracting telemetry measurements.
- Data storage options include PostgreSQL and a hybrid mode with Cassandra for scalability.
- Open-source with a free version available.
- Scalable for monitoring from individual devices to large-scale deployments.
- Compatible with heterogeneous hardware and software components.
- Provides real-time device telemetry monitoring.
- Supports simulated data for testing and experimentation.

The ThingsBoard architecture consists of different components as specified and described below [96]:

- The ThingsBoard Transport component is responsible for receiving the messages from the sensors and devices in the network using communication protocols, parsing the message, and pushing it into the message queues.
- ThingsBoard Core handles REST API calls, WebSocket subscriptions on entity telemetry and attribute changes, stores the information about device sessions, and monitors the device connectivity state.
- The ThingsBoard Rule Engine is considered the main component of the system, and it subscribes to incoming messages and processes them. This component uses Node-Red for creating the rules and flows.
- The ThingsBoard web UI that is stateless and is written in Express.js.

We created this smart city prototype to gain insight into several questions about the management activities in a smart city:

- What aspects of a smart city need monitoring and management?

- How can we monitor and manage the resources in a smart city infrastructure to deal with challenges such as the huge amount of data?
- Which features should an autonomic management system include?
- What operational actions could be done by an autonomic management system?

In this prototype, we explore the structure to enable components to collect the telemetry data from the sensors deployed in the smart city, categorize, and visualize them and provide a real-time view of what is going on in the city. Telemetry data is composed of the measurement information about the environment in which the sensors are deployed and sometimes the details about the sensors and devices themselves. For our prototype, we assumed that our smart city contains several sensor objects whose data are collected, logged and stored in a dataset or data files.

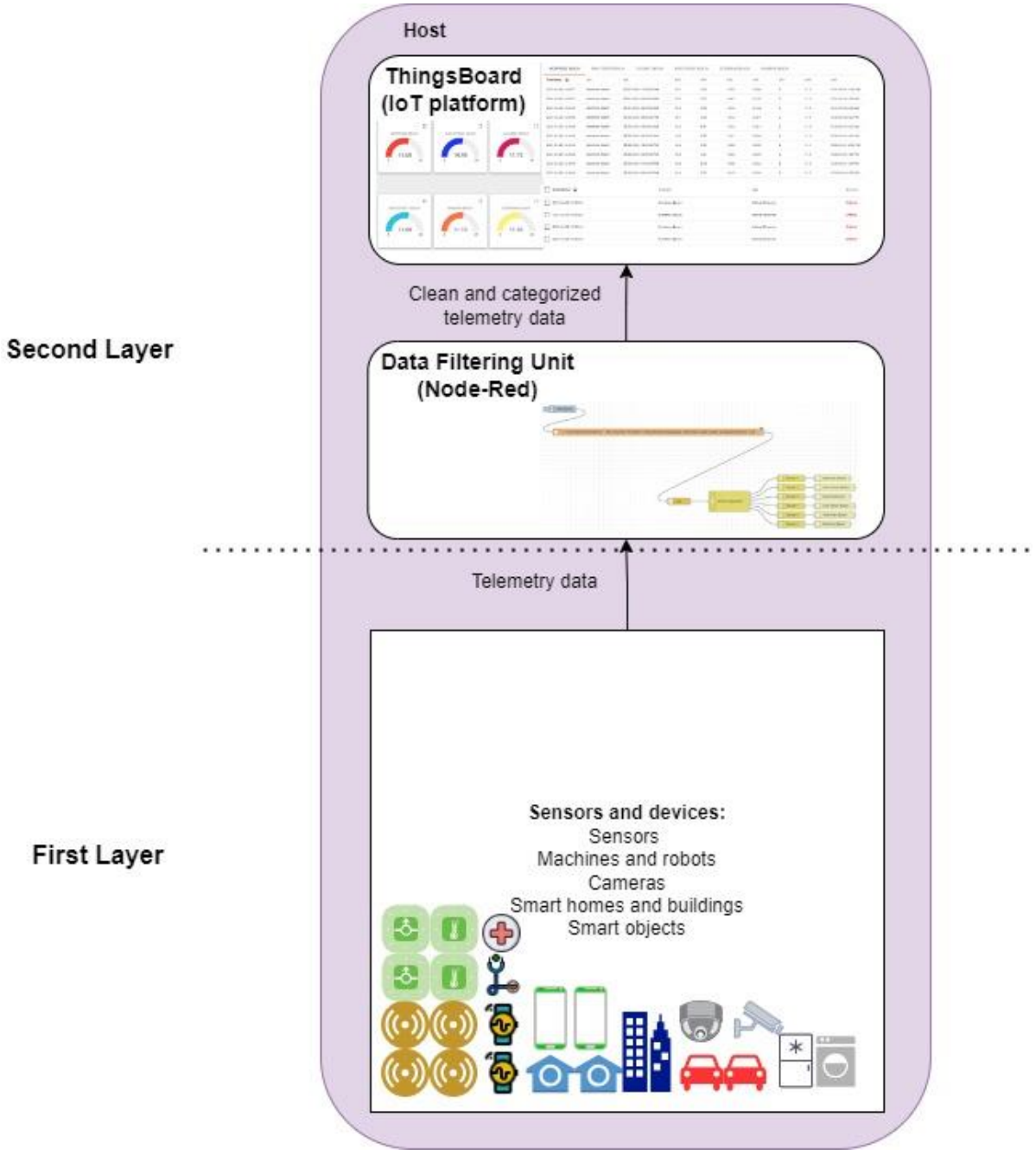


Figure 5.1 Smart City Prototype

For experimentation purposes, we use data collected from sensors used to monitor the quality of water deployed at Chicago beaches [97]. This dataset provides information about beach-name, water temperature, turbidity, Transducer-depth, wavelength, wave-period, and battery life and contains more than 34,900 records. The chosen version of the IoT platform lacks

certain tools that are essential for classifying data in its rule engine. To address this limitation, we use Node-Red as our data filtering unit. This unit is responsible for separating and sorting the incoming data based on specific criteria. The data filtering unit receives information from various sensor objects through communication protocols. These sensors collect data from the environment and send it to the filtering unit for processing. By using this approach, we can ensure that the data received by our IoT platform is properly organized and categorized, which allows us to make better decisions based on that data. In Node-Red, a flow is created that starts with reading our sensor data from the dataset and clustering and filtering the data based on the sensor name using the switch node. Then each stream of data is guided to the corresponding output. After that, this output is sent to the related sensor in ThingsBoard using the HTTP protocol. To assign the Node-Red output to the correct device in ThingsBoard, the device access token is used that can be obtained from the ThingsBoard device details tab. Finally, the ThingsBoard IoT Platform is utilized to provide data visualization, analyze the data, and trigger alarms based on data fluctuations; The smart city prototype and the experiments conducted with it have been documented in a published paper [94].

In our prototype, we have designed an asset for each beach to be monitored. An asset is essentially a specific entity, such as a building or a house, which has been equipped with various sensors and can be managed using the IoT platform, ThingsBoard. Once an asset has been created, we then add a set of sensors to the device management section and assign them to the corresponding asset. Then each sensor in ThingsBoard receives the telemetry data that belongs to it and this telemetry data is then displayed in the latest telemetry tab, allowing users to easily access and analyze the data in real-time. One of the key benefits of ThingsBoard is its ability to handle multiple streams of telemetry data simultaneously. This means that even if there are multiple sensors sending data at once, the platform is able to manage the data and display it all within a single, easy-to-use interface. ThingsBoard offers a wide variety of widgets to create and customize a dashboard to visualize the data in real-time. It is also possible to develop a widget using HTML, CSS, and JavaScript in its built-in IDE. In the next section, some experiments are described along with visualization of data.

To summarize, the smart city prototype contains the following components:

Layer 1 – Devices and sensors

This layer includes all the devices, sensors, and other things that are connected in a smart city and produce data in real-time. For our initial experiments, we simulate the smart city sensors using a dataset that had sensory data about the quality of water at Chicago beaches [97] and provide a dashboard that uses the data as in our simulated smart city. This data is raw and has some redundant or unnecessary information that is not useful for our simulated smart city environment. It needs to be cleaned for our use in our simulated smart city environment.

Layer 2- Data filtering and cleaning (Node-Red)

The data filtering and categorization in our architecture are done by Node-Red. There are two reasons for adding this component. First, as is mentioned in the previous section the data is not cleaned, so we needed to make the data clean before entering and populating the database to save the resources which is very critical in the smart city environment where we deal with huge amount of data and limited resources. Also, our dataset includes all sensors' data, and it is not categorized, so the data should be categorized based on the sensor number or name and each subset of data should go to the corresponding sensor in the ThingsBoard device section. Therefore, in this step, a flow is created to categorize the data. This flow starts by getting the data set and then a "switch" node is added for categorization purposes. In this node, the data classification is done using the sensor name. Then each stream of the data gets cleaned, filtered and navigated to the corresponding output and at last each output will be directed to the corresponding sensor or device in ThingsBoard using the POST method in HTTP. In this layer, it is also possible to apply machine learning techniques to the data.

Layer 2 – Device monitoring and data visualization (ThingsBoard):

The first layer, devices and sensors, is monitored using our selected IoT platform, which is ThingsBoard. ThingsBoard can operate over a long period of time and collect telemetry data and monitor them. On this side, after creating an asset for each building or region, and their assigned devices and sensors, the ThingsBoard is fed with the data coming from the previous layer. In this step, we are able to see the latest telemetry data for each device or sensor. Next, in order to perform the data visualization and monitoring task we have to make a dashboard.

Each dashboard includes some widgets that are responsible for the representation of the data in different formats such as tables, charts etc. In this step, it is also possible to set alarms based on the data fluctuations and notify the city authorities about critical events.

5.1.2 Experiments with Smart City Prototype

In this section, we present experiments done using ThingsBoard and our smart city prototype. We start with the representation of the state of one of the sensor attributes for all the deployed sensors. Then we continue by providing all the data for each sensor respectively. Lastly, we configure ThingsBoard to create an alarm when a certain condition is met. Our dataset is taken from [97] which consists of the sensory data coming from the sensors deployed in the water at several beaches in Chicago.

5.1.2.1 Dashboard to Show the State of One Attribute for all Sensors.

In our first experiment, we will focus on just one attribute that is common among all the sensors. Several sensors are added to the ThingsBoard in our prototype, and although these sensors have different functions, they all have a common attribute that needs monitoring - their battery life. Battery life is crucial to ensure the uninterrupted functioning of the sensors. The dashboard has been designed to include a digital gauge that represents the battery life of each sensor. Each sensor has its own gauge on the dashboard, ensuring that the battery life of each sensor is monitored individually in real-time. This feature allows for efficient tracking and management of the battery life of each sensor. For instance, Fig. 5.2 provides a clear visual representation of the battery life of all the sensors. and the administrator can check them regularly to ensure that they are in good condition. Apart from monitoring the battery levels in real-time, alarms can be set up to notify the administrator when the battery life goes below a particular threshold. This ensures that the administrator is alerted well in time and can take necessary steps to prevent any disruption in the functioning of the sensors. In the next section, we will talk about the alarms and how they are shown on the ThingsBoard dashboard.

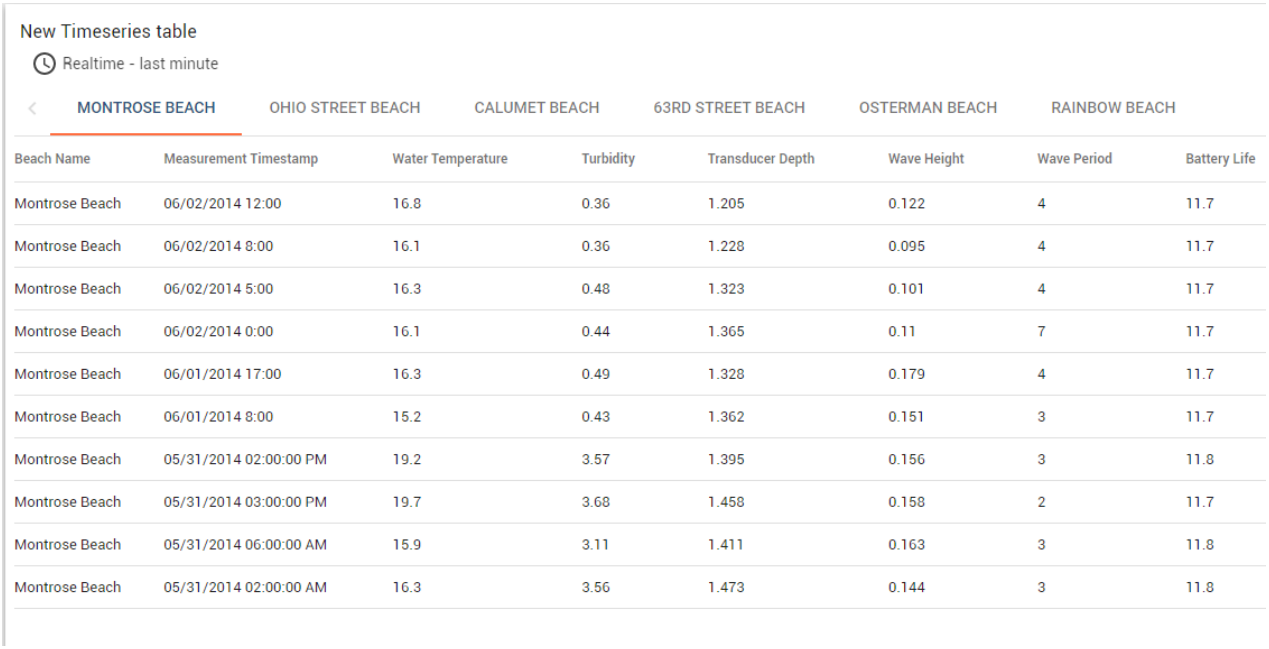
Figure 5.2 ThingsBoard dashboard to show the battery life of the sensors.

5.1.2.2 Visualization of the Telemetry Data by Sensor Location

In our second experiment, we will aim to present the data coming from all the sensors deployed on the beaches. A separate dashboard is created for each asset. This dashboard presents the data collected by the sensors that are assigned to that particular asset.

To present and visualize the data, a time-series table widget is used in the ThingsBoard dashboard. This widget enables us to extract telemetry data from the sensors and show it in a table format that is organized in separate sections split by tabs with the sensor location. This allows us to quickly and easily view the data for a particular sensor location. Fig. 5.3 provides an example of how this looks.

Once the Node-Red flow is initiated, the data records start appearing one after the other in the dashboard. This feature allows us to monitor the data in real-time and observe any trends or patterns.



New Timeseries table

Realtime - last minute

MONTROSE BEACH | OHIO STREET BEACH | CALUMET BEACH | 63RD STREET BEACH | OSTERMAN BEACH | RAINBOW BEACH

Beach Name	Measurement Timestamp	Water Temperature	Turbidity	Transducer Depth	Wave Height	Wave Period	Battery Life
Montrose Beach	06/02/2014 12:00	16.8	0.36	1.205	0.122	4	11.7
Montrose Beach	06/02/2014 8:00	16.1	0.36	1.228	0.095	4	11.7
Montrose Beach	06/02/2014 5:00	16.3	0.48	1.323	0.101	4	11.7
Montrose Beach	06/02/2014 0:00	16.1	0.44	1.365	0.11	7	11.7
Montrose Beach	06/01/2014 17:00	16.3	0.49	1.328	0.179	4	11.7
Montrose Beach	06/01/2014 8:00	15.2	0.43	1.362	0.151	3	11.7
Montrose Beach	05/31/2014 02:00:00 PM	19.2	3.57	1.395	0.156	3	11.8
Montrose Beach	05/31/2014 03:00:00 PM	19.7	3.68	1.458	0.158	2	11.7
Montrose Beach	05/31/2014 06:00:00 AM	15.9	3.11	1.411	0.163	3	11.8
Montrose Beach	05/31/2014 02:00:00 AM	16.3	3.56	1.473	0.144	3	11.8

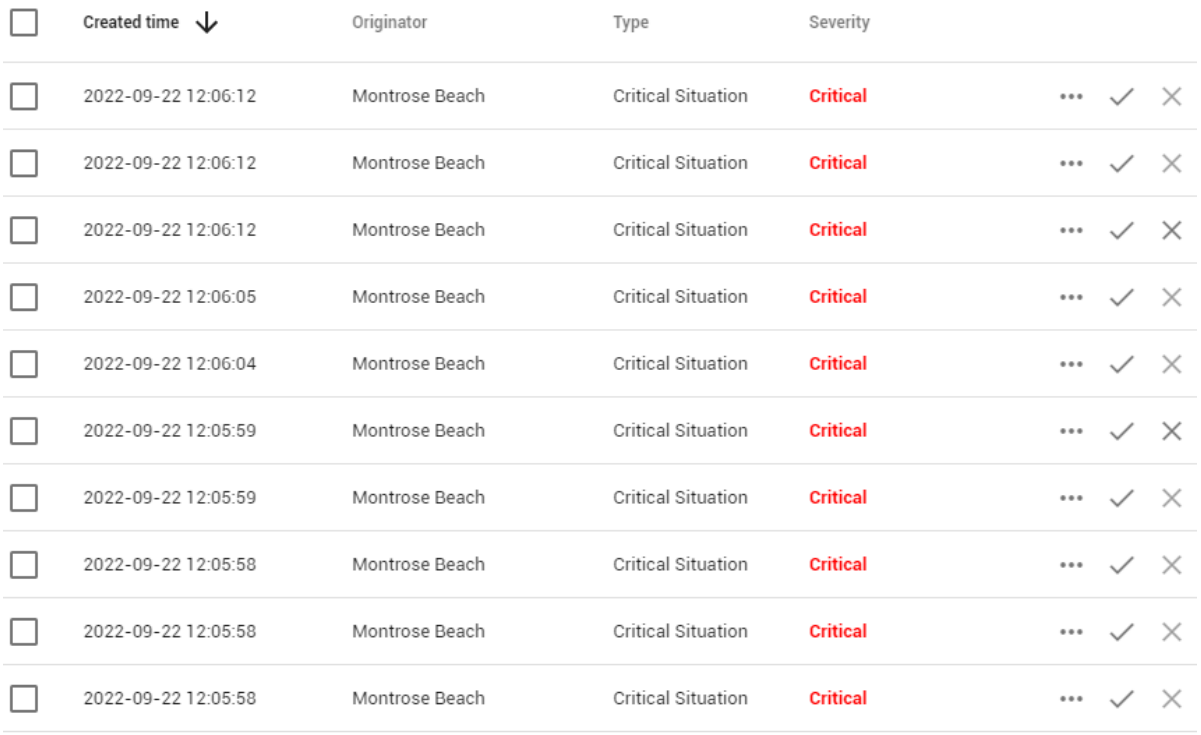
Figure 5.3 ThingsBoard time-series dashboard to show sensor telemetry data.

5.1.2.3 Triggering Alarms and Generating Notifications.

After collecting and analyzing the data from the sensors on the beaches, we create alarms based on sudden changes in the data. For instance, we have a sensor placed on each beach to measure wave height. By monitoring this sensor, we can create alarms to alert people living near the sea to be prepared for a possible tsunami or to evacuate their homes in the event of a catastrophic situation. Alarms in ThingsBoard are manually set by the administrator and are limited in their expressibility.

In this use case, we set a limit for the wave height. When the height of the waves exceeds this limit, an alarm is triggered, providing the time and severity of the situation. This enables people to quickly respond and take appropriate measures to ensure their safety.

To help manage these alarms, we created an alarm dashboard that displays the relevant information in real-time. This dashboard presents the alarms created for each beach and enables us to track them efficiently and respond quickly if any emergency occurs. Fig. 5.4 provides an example of how this looks.



Created time	Originator	Type	Severity
2022-09-22 12:06:12	Montrose Beach	Critical Situation	Critical
2022-09-22 12:06:12	Montrose Beach	Critical Situation	Critical
2022-09-22 12:06:12	Montrose Beach	Critical Situation	Critical
2022-09-22 12:06:05	Montrose Beach	Critical Situation	Critical
2022-09-22 12:06:04	Montrose Beach	Critical Situation	Critical
2022-09-22 12:05:59	Montrose Beach	Critical Situation	Critical
2022-09-22 12:05:59	Montrose Beach	Critical Situation	Critical
2022-09-22 12:05:58	Montrose Beach	Critical Situation	Critical
2022-09-22 12:05:58	Montrose Beach	Critical Situation	Critical
2022-09-22 12:05:58	Montrose Beach	Critical Situation	Critical

Figure 5.4 ThingsBoard alarm dashboard

5.1.3 Discussion on the Smart City Prototype

The development of the prototype and experiments allowed us to better understand the entire process of IoT data collection, analysis, and management. In developing the smart city prototype and performing the experiments we could get a sense of how we can connect the devices and sensors to the IoT platform, how to categorize the data collected from these devices and sensors, and how to send and receive telemetry data effectively. Moreover, we were able to visualize this data using dashboards, trigger alarms based on predefined limits, and how to filter the data, making it easier to analyze and interpret [94].

As noted, we also wanted to use this prototype to help understand several questions pertaining to the autonomic management of a smart city. Some of the insight gained is outlined below:

- What aspects of a smart city need monitoring and management?
    - Other than the sensors and devices that we knew already needed management, we found out that network and infrastructure performance and other resources need monitoring and management.

- The use of an IoT platform, ThingsBoard in our case, provides useful capabilities for managing aspects of sensors and collecting data about the sensors. While such a platform can be useful with the monitoring and management of a smart city, it must be integrated into the overarching autonomic management system to ensure a seamless management environment.

- How can we monitor and manage the resources in a smart city infrastructure to deal with challenges such as the huge amount of data?
  - In a smart city infrastructure, monitoring and managing the resources can be a challenging task due to the large amount of data generated by smart city elements and components. To effectively deal with this challenge, it is important to leverage autonomic management techniques.
  - To monitor the resources, instrumentation and monitoring tools can be employed to collect and analyze data from various smart city components. This data can be used to create a comprehensive view of the resources and their utilization in the smart city infrastructure. While this prototype did not include a component to collect data about the infrastructure, our experience with ThingsBoard indicates that such a component would also need to be integrated with the autonomic management system; this is discussed more in the next section.
  - To manage the resources, policies can be defined and enforced to address any violations or issues that arise. These policies should be triggered automatically when certain conditions are met, allowing for proactive management of the resources.
  - It is important to note that this monitoring and management should be autonomic in nature. A smart city infrastructure operates 24/7 and relying on a single administrator or administrative team for monitoring and management is not optimal and efficient. Autonomic management can enable the infrastructure to self-manage, self-heal, and optimize its performance based on changing conditions and demands, thus increasing its efficiency and reliability.

- Which features or components should an autonomic management system include?
    - Before building and conducting experiments with the prototype, we had a general idea of the components required for the autonomic management system, such as data reception from the smart city infrastructure, and a decision-making component. However, during the implementation of the prototype, we realized the necessity to store policies and information about thresholds and desired metric values. We also discovered that not only is the real-time monitoring of the smart city ecosystem crucial, but in some cases, real-time management is also necessary. Furthermore, we recognized the importance of establishing communication and collaboration among the system components to ensure effective coordination. Moreover, the realization of a feedback loop became evident as a means to continually monitor the smart city, perform tasks, and ensure conditions are met. These insights shaped the development of our autonomic management system.
- What operational actions could be done by an autonomic management system?
  Although we had some ideas about some of the functionality of the Autonomic Smart City Management System such as configuration of the smart city ecosystem, from the prototype we could find out that an ideal autonomic smart city management system should include the below features:
    - Optimization: The system should be able to optimize the smart city infrastructure performance
    - Healing: The system should be able to identify and repair any issues or faults that arise in the smart city infrastructure without human intervention.
    - Policy-based management: The system should allow policies to be defined and enforced to ensure compliance with regulations and standards.
    - Resource monitoring: The system should be able to monitor resources dynamically and manage them.
    - Performance monitoring: The system should be able to monitor the performance of the infrastructure and detect any anomalies or deviations from expected behavior.

- Fault detection and diagnosis: The system should be able to detect and diagnose faults or issues in the infrastructure and take appropriate action.
- Reporting: The system should be able to provide reporting capabilities to provide the administrators with a clear overview of the smart city environment and infrastructure, along with detailed insights into the outcomes of the actions executed within the smart city.

By including these features in an autonomic management system, a smart city infrastructure can operate more efficiently, and reliably, while minimizing the need for human intervention.

5.2 Autonomic Management System Prototype

The autonomic management prototype is composed of several components as shown in Fig. 5.5 that are built on top of the smart city prototype discussed in the previous section. These components work together to create a fully functional autonomic management system. The key components of the autonomic management prototype are as described in Chapter 4, these are:

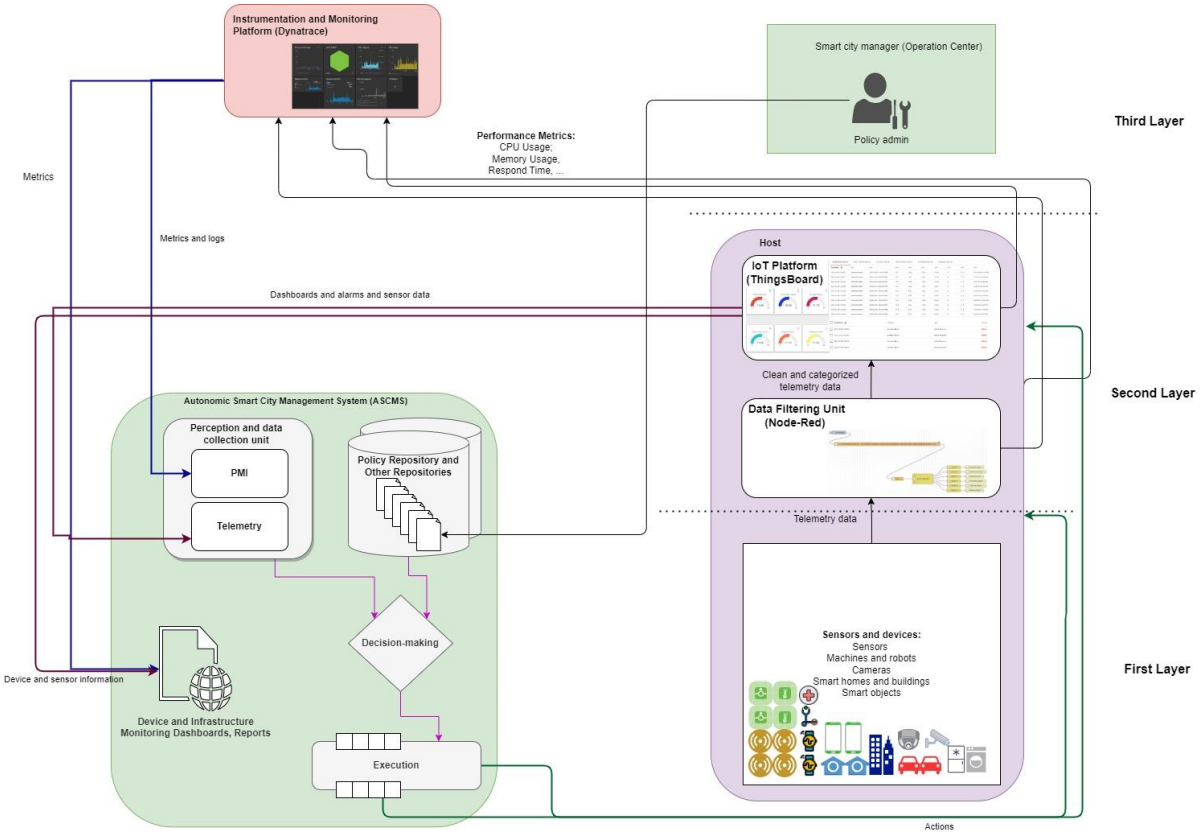


Figure 5.5 Autonomic Smart City Management System Prototype

Perception and data collection unit: This layer is responsible for receiving data from multiple resources in the smart city ecosystem.

Performance monitoring and instrumentation unit: This layer can be a part of the autonomic management system or can be an external unit. In our prototype this layer is outside the system. The responsibility of this layer is to monitor the performance of the smart city infrastructure such as CPU, memory usage and so on.

Policy repository: In this component all the policies are stored to be utilized for the decision-making process. The policies define the rules and conditions under which certain actions are taken.

Decision making unit: In this unit the collected data is analyzed, and the appropriate action is determined according to the stored policies.

Execution unit: This unit is responsible for executing the operational actions decided by the decision-making unit.

The perception and data collection unit is responsible for collecting two types of data. The first type is called telemetry data, which comes from the IoT platform using the MQTT protocol. The second type is called performance monitoring data, and it comes from performance monitoring and instrumentation unit. In the previous section, we talked about the IoT platform. In this section, we'll discuss the monitoring and instrumentation platform, and we'll explain about the specific instrumentation platform we've chosen, which is called Dynatrace.

The core algorithm of the Autonomic Smart City Management System is provided below:

```
1. operational <- True
2. while operational:
3.         M <- Get_Metrics_Telemetry
4.         I <- Get-Info
5.         RR <- Get_Rules
6.         R <- Check_Rules(RR)
7.         for each rule in R:
8.               A <- Extract_Actions(rule)
9.               result <- Execution_Component(rule,A)
10.              if not result:
11.                    report(A,rule)
```

Each line of the algorithm is described in more detail as follows:

1. Initiates the looping mechanism by setting a Boolean variable until it becomes false at some future point.
2. Loop starts.
3. The metrics and telemetry data are received by the “Perception and Data Collection Unit.”
4. Essential information required for decision-making, such as thresholds, is retrieved from the repository.
5. The "Get-Rules" function retrieves rules from the "Policy Repository."

6. “Check_Rules” evaluates the condition of policies in the “Decision Making Unit”. It returns a set of policies that have been violated or an empty set if none are violated.
7. If there are policies that have been violated, each one is processed individually, and corresponding actions are carried out.
8. Get action or actions associated with the specific policy.
9. Invoke the Execution_Component to carry out the actions associated with the policy. If all actions are successfully executed, the Execution_Component returns True.
10. If an action fails, a message is printed about which policy and actions failed.

5.2.1 Monitoring and Instrumentation Framework

In the previous section, the importance of monitoring and visualization of the sensor and device data, called telemetry data, was identified as essential in a smart city. But, as noted, in a smart city just collecting telemetry data is not sufficient - we also need to be able to collect data for the management of the smart city infrastructure. There are many aspects to the management of smart city infrastructure; our research concentrates on not only the telemetry data and sensor attributes but the operational aspects of the architecture and aims to extract metrics, particularly performance metrics, from the smart city components. To extract those metrics a monitoring platform is used. This platform provides observability and measurement information about the smart city infrastructure through metrics, logs, and traces. Some of the measurement metrics that can be obtained from the monitoring platform include response times, throughput, memory and CPU usage, etc. By monitoring these metrics, we can gain insight into how the smart city infrastructure is performing and identify any areas that may need improvement. For instance, if response times are slower than usual, we can investigate the cause and take action to improve performance. Additionally, the monitoring platform allows us to monitor each service and process individually, giving us a more granular view of the smart city's performance.

To build our prototype, we selected Dynatrace as our monitoring and instrumentation platform. It allows us to keep an eye on all the components, processes, and services of the infrastructure, including Node-Red and ThingsBoard, and gather their performance metrics.

Because we consider extracting the performance metrics from the infrastructure, we first need to set up the monitoring agents on the infrastructure host to monitor its performance metrics, and then we can view the metrics in the dashboard. Based on [98] the formal definition of host is “computers that provide certain services or resources within a network that other participants within the network can then access and use.”. This host can be a cloud or local server. In our prototype, this host is a single machine where all the smart city infrastructure components are deployed. Fig. 5.6 to 5.11 present the dashboards and statistics that the Dynatrace instrumentation platform provides for Node-Red and ThingsBoard processes and services.

Fig. 5.6 showcases a comprehensive dashboard illustrating the performance metrics of the Node-Red service. This dashboard offers insights into critical indicators such as response time, CPU usage, failure rate, and throughput. Additionally, it provides valuable information regarding the utilization of applications, databases, and other resources utilized by Node-Red. On the other hand, Fig. 5.7 represents the general Node-Red dashboard, encompassing a broader range of properties and tags. This dashboard offers specific details surrounding the Node-Red process. It provides a comprehensive and granular perspective on the functionality and performance of Node-Red, enabling a deeper understanding of its inner workings and capabilities.

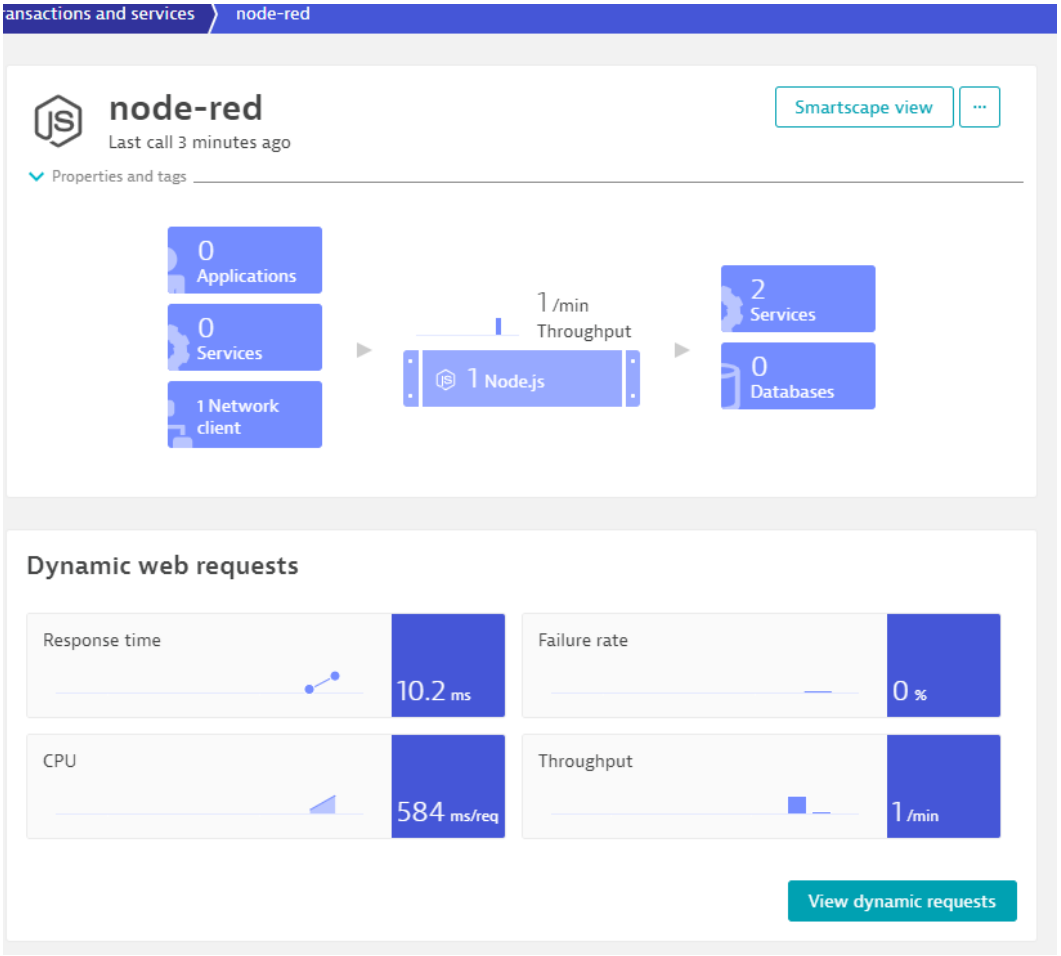


Figure 5.6 Node-Red service Performance Metrics

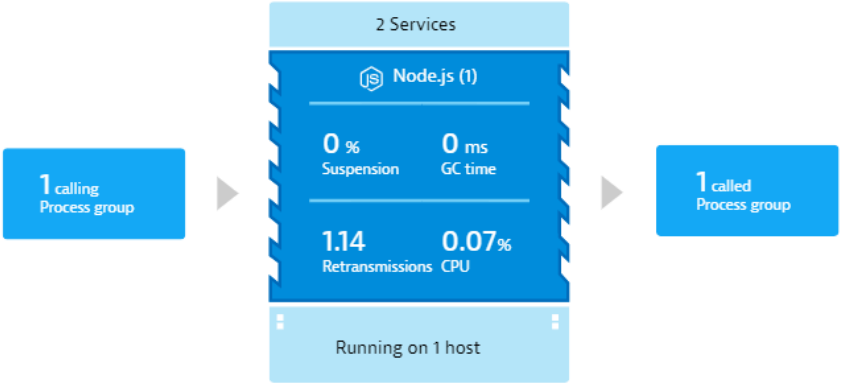


Figure 5.7 Node-Red Dashboard

Fig. 5.8 and Fig 5.10 present dashboards showcasing the performance metrics of the Thingsboard (SpringBoot) and ThingsBoard database (PostgreSQL) services. These dashboards provide insights into crucial indicators, including response time, failure rate, and throughput. These dashboards also have information about the components and data flow of the services.

Fig. 5.9 and 5.11 feature the general Thingsboard (SpringBoot) and Thingsboard (PostgreSQL) dashboards, offering a range of properties and tags. These dashboards delve into the intricate technologies and specific details associated with these two processes, providing a comprehensive and detailed perspective on their functionality, performance and services that ThingsBoard processes are using. They enable a deeper understanding of the inner workings and capabilities of Thingsboard (SpringBoot) and Thingsboard (PostgreSQL).

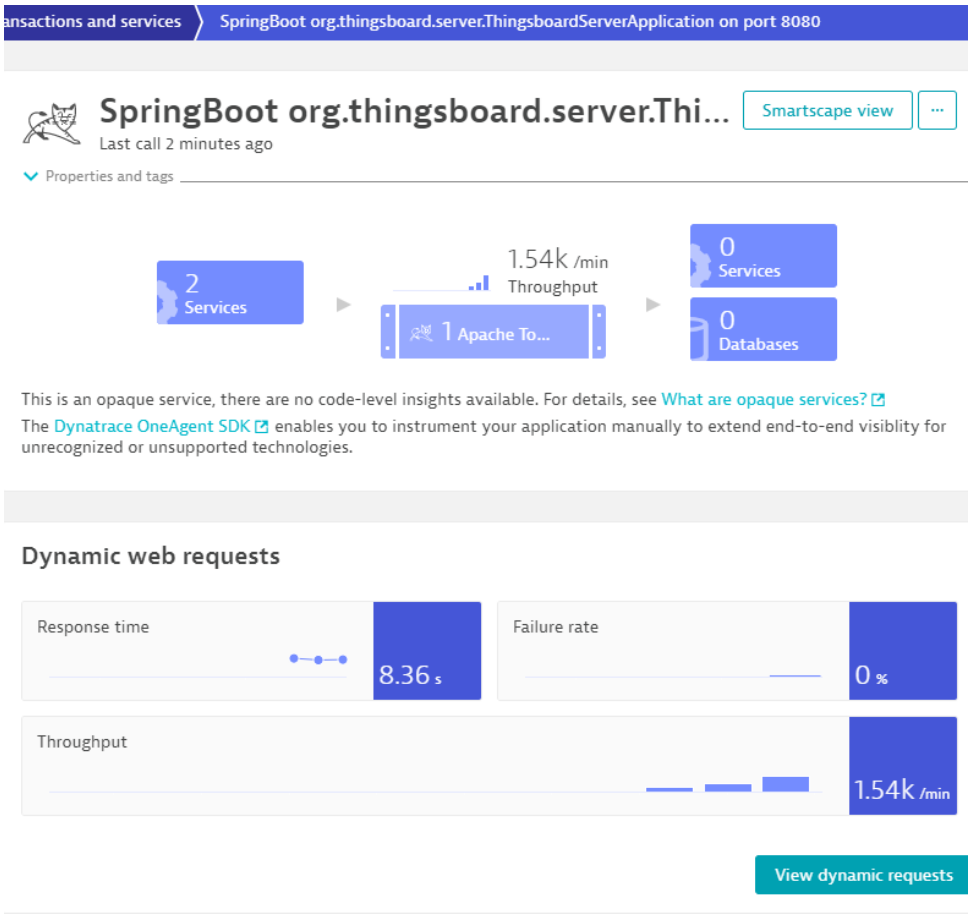


Figure 5.8 ThingsBoard (SpringBoot) Service Performance Metrics

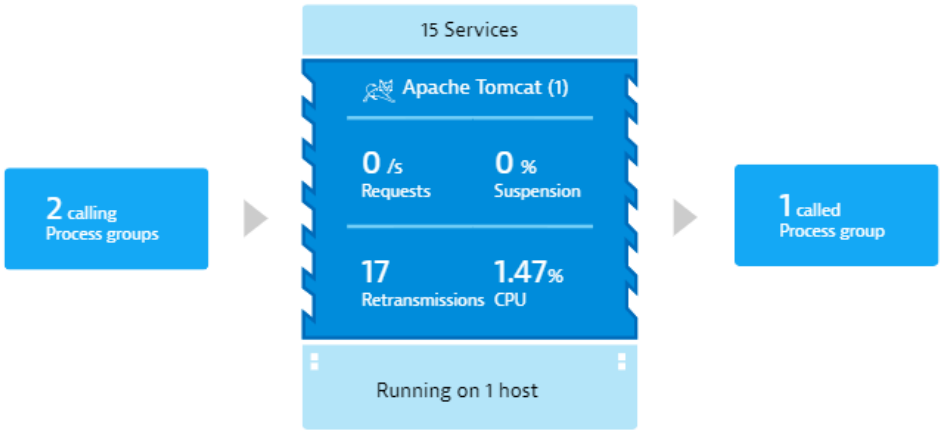


Figure 5.9 ThingsBoard (Spring boot) Dashboard

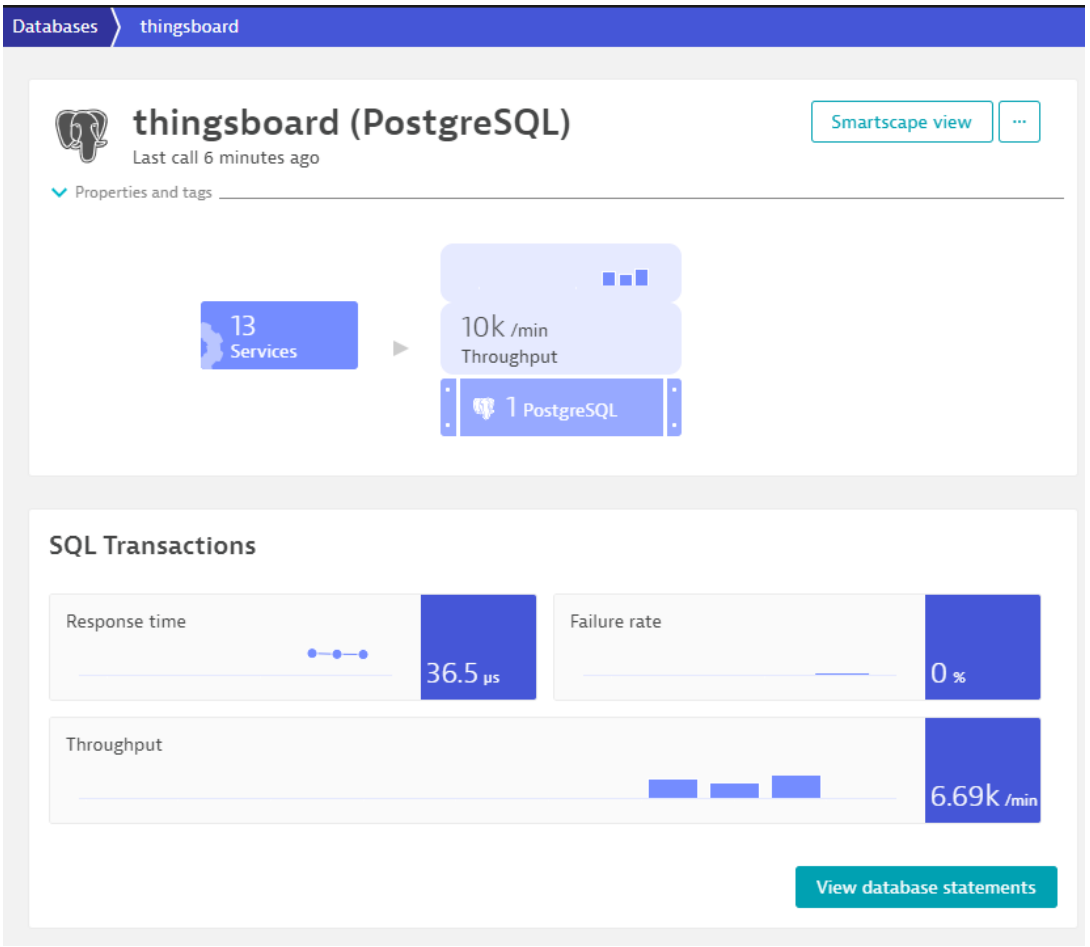


Figure 5.10 ThingsBoard (Database) Service Performance Metrics

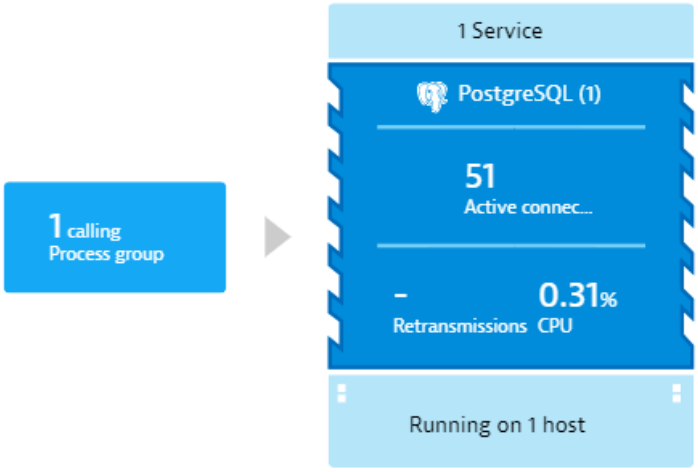


Figure 5.11 ThingsBoard (Database) Dashboard

Dynatrace is a commercial tool that provides extensive instrumentation and monitoring capabilities. Like ThingsBoard, it provides key functionality needed for the management of applications and services in a smart city. We must, however, ensure that it is integrated into the overall operation of the autonomic management system.

5.2.2 Device and Infrastructure Monitoring Web Platform

The device and infrastructure monitoring web platform is the optional component of the autonomic system, but we implemented it as an informational interface to provide metrics and data from ThingsBoard and Dynatrace in one place. Fig. 5.12 is the web platform for our autonomic smart city management system that is composed of several parts:

The first component is the dashboard, which is accessible from the homepage and provides real-time updates on the performance metrics. This dashboard serves as a central hub where administrators can quickly assess the system's status.

The device management page is the next page of our web platform, which is seamlessly integrated with our IoT platform. This page provides administrators with access to a wealth of information, including alarms, dashboards, and streaming data from the devices and sensors deployed throughout the smart city.

The next page is the infrastructure monitoring page which provides detailed insights into the performance metrics of the smart city infrastructure, presenting the metrics and their corresponding values in an easy-to-understand format.

The last tab of the web platform is the infrastructure management which provides administrators with a comprehensive view of the results of any management actions taken within the system. Additionally, this page can present reports and notifications, allowing administrators to stay informed about the autonomic management system functionality. The management section has access to the policy engine and the result of the corrective actions. Upon successful completion of any corrective action, this page will promptly publish the results to notify administrators.

The web platform is meant to serve as an informational platform for administrators and city authorities, intended to provide information rather than manage the smart city ecosystem. It provides them with valuable insights into how the autonomic smart city management system is functioning. If the metric is based on the condition stored in the policy engine, the corresponding action is planned and executed in the smart city ecosystem to ensure that the system operates in accordance with the high-level goals set by the administrators. To keep the administrators informed, every time a self-configuration takes place in the smart city ecosystem, a notification text appears on the screen. This notification acts as an alert mechanism, giving the administrators an indication that all components, the environment, and the whole infrastructure are under control.

That being stated, the addition of pertinent capabilities opens the door for potential expansion, allowing for the integration of the management interface aspect, effectively serving as a management interface. It is worth noting that in our work the Autonomic Smart City Management System remains responsible for the overall management of the smart city and not the web platform.

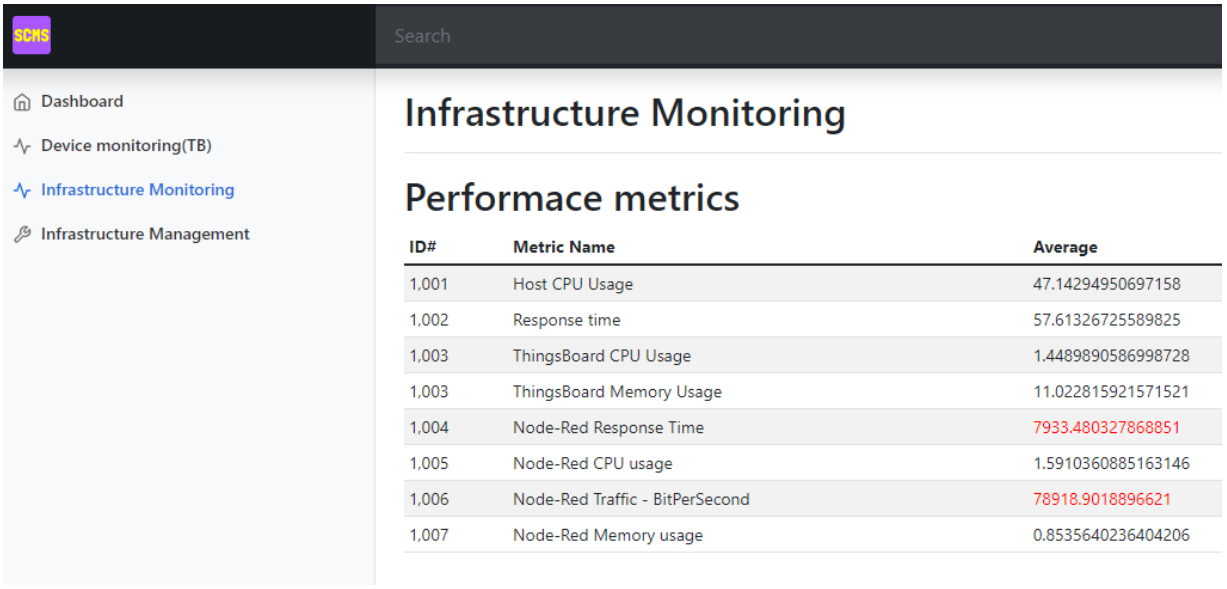


Figure 5.12 Device and Infrastructure Monitoring Web Platform

5.2.3 Management Policies

This section presents the policies that we defined to manage the smart city environment and its infrastructure. For each event in a smart city, one or several policies have been defined and the proper action or actions are determined to deal with the problem or to adapt to the change.

To make it easier for the administrator to add the policies to the database, we created a form that provides a graphic interface for adding the policies as shown in Fig. 5.13.

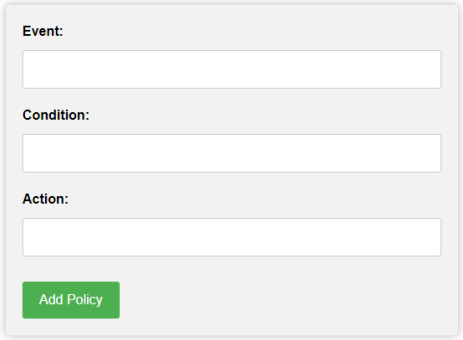


Figure 5.13 Policy Form

The policies are stored in a robust SQLite3 database, ensuring efficient access and retrieval. Also, JSON is used for data handling and parsing the payload. The policy language serves as the framework for defining domain-specific policies. Within this language, each policy aligns with a distinct domain or area of concern. These policies are structured by a collection of rules, wherein each rule precisely articulates both a condition and a corresponding action. When the specified condition is satisfied, the associated action is executed.

5.2.3.1 Sample Policy

For example, one policy is defined as below:

```
Sensor_Battery_Full:
On Battery_full(s)
      If battery_level(s) == 100
      Then SwitchtoBattery(s)
```

Policy Name: "***`Sensor_Battery_Full`***" is the name of the policy, which is a user-defined identifier for this specific rule.

Event Trigger: "`On Battery_full(s)`" is the event trigger that specifies when this policy should be executed. In this case, the policy is triggered when the battery level reaches 100%. In this case, the policy evaluation component would check for data from ThingsBoard on battery levels.

Condition: "`if battery_level(s) == 100`" is the condition that determines whether the policy's action should be executed. It checks if the battery level of a particular sensor (represented by '`s`') is equal to 100%. If this condition is met, the action will be performed.

Action: "`then SwitchtoBattery(s)`" is the action that specifies what should happen when the condition is met. It indicates that the system should switch to battery power for the sensor represented by '`s`' when the battery level reaches 100%. For the simulation, we have implemented functions to simulate the real-world effects of each action that the prototype ASCMS could take. For instance, when executing the "`SwitchtoBattery(s)`" action, we have designed a function that changes the power source to the battery, while also gradually

depleting the battery level to mimic the authentic behavior. However, sometimes in instances such as updating firmware versions, the administrator may be required to write code and integrate it into the repository. Note that such code, once written and added to the repository, can be used for updating firmware for the same type of sensors or even other similar sensors.

As the core process of the autonomic management system executes, it dynamically retrieves attributes, evaluates the conditions, and triggers the appropriate actions accordingly. Following each iteration, the policy checker process momentarily pauses, allowing for a designated period of time to elapse. Subsequently, it resumes its task of evaluating the prevailing conditions to determine whether the activation of policies is warranted once more. This deliberate pause-and-resume approach ensures that our policies are consistently monitored and applied in a judicious manner. Our prototype system encompasses 12 distinct scenarios, for every scenario, its specific policy or policies are stored in the database. We provide an overview of the scenarios and policies in the following.

5.2.3.2 Sensor Firmware Version

There are several reasons for updating the sensor's firmware: a) To improve the sensor functionality and accuracy; b) To fix issues and bugs; c) To protect it against security attacks; and d) To add new features to the sensor. For doing so, we first need to get the sensor's current version from the IoT Platform and compare it with the latest version. In case the sensor firmware version is outdated, the smart city management system needs to send a message to the IoT Platform that the sensor needs to be updated.

```
Sensor Version:
On NewFirmwareVersion(s)
      If firmwareVersion(s)! = latestVersion(s)
      Then UpdateSensorVersion(s)
```

`NewFirmwareVersion(s)` is the detection of an event indicating the release of a new version of the firmware associated with a sensor "s". This would be triggered when the administrator adds a version to the repository. The functions `firmwareVersion(s)` and `latestVersion(s)` are predicates that get the current version of the firmware for sensor `s`

and the latest firmware version for the sensor. Then `UpdateSensorVersion(s)` would instruct the IoT platform to update the firmware of sensor `s`. Note that the policy describes the condition for ANY sensor – as long as it has a firmware version (in practice – some sensors maybe very simple and not have any firmware – just hardware)

5.2.3.3 Offline Sensor

Occasionally, the sensor may go offline due to internal errors or a drained battery. To address such situations effectively, we follow a specific protocol. The "Sensor Offline" policy consists of three rules that form the foundation for managing the status of sensors, whether they are offline or online. Firstly, we attempt to resolve the issue by rebooting the sensor. If this step fails to restore connectivity and bring the sensor back into the network, it is likely that the battery needs recharging. Consequently, the power supply is switched to direct current (DC) to facilitate the charging process. However, if switching to DC power does not resolve the problem, the (ASCMS) triggers an application to promptly notify the operator. The `lastConnectTime` is the data that is received from ThingsBoard.

The first solution when the sensor is offline `(lastConnectTime(s)<currentTime)` is that the ASCMS triggers a reboot operation. In order to address this, we defined the following policy:

Sensor Offline:

```
On SensorOffline(s)
      If lastConnectTime(s)<currentTime
      Then Reboot(s)
```

in the above policy, `Reboot ()` would send a message to the IoT Platform to reboot the specific sensor AND would set the `reboot` flag to True.

To address the situation where a sensor has been recently rebooted, a separate rule comes into play. If a `sensor(s)` has been rebooted (`reboot(s)==True`) and possesses a DC capability (`HasDC(s)`), the autonomic management system initiates a message to the IoT Platform, instructing it to switch the sensor to DC power mode (`SwitchToDC(s)`) and changes the `powerSource` attribute to DC.

```
On SensorOffline(s)
    If lastConnectTime(s)<currentTime and reboot(s)==True and
    HasDC(s)
    Then SwitchToDC(s)
```

Here, `reboot(s)==True` checks flag `reboot` of the sensor `s` to see if the sensor `s` was rebooted and if it has a DC capability then the Autonomic management system would send a message to the IoT platform to try to switch that sensor to DC.

Then if none of the above solutions were helpful meaning that `reboot` flag is True and `powerSource` is DC but still the sensor is offline the system should initiate an application to notify the operator.

```
On SensorOffline(s)
    If lastConnectTime(s)<currentTime and reboot(s)==True and
    powerSource(s)= "DC"
    Then NotifyOperator ()
```

So here is where the management system is generating a notification to seek human intervention.

5.2.3.4 Response Time

Based on [99] the response time below 10ms is acceptable but if the response time is constantly more than 10ms we need to be careful. If the waiting time for the storage takes more than 50 ms, it should be taken very seriously. Therefore, we created the following policy to deal with this problem. The response time could be for a service, a process or an application. In this case we focus on the response time of an application. If the response time goes higher than 10ms we add compute nodes to decrease the response time and increase the overall performance. For simplicity, this approach uses vertical reactive scaling. It is vertical meaning that we add more resources to the service and also it is reactive which means when the mitigation takes place after the response time increased. The key limitation of reactive scaling is that after the system encounters the overload the management system can manage the resources and decrease the response time. To save resources and utilize them efficiently we can decrease the compute nodes if the workload is not massive, and the response time is acceptable (below 10ms).

```
Check_Response_Time:
On True
      If responseTime(app)> 10000
      Then addComputeNodes (app)
```

An event "`True`" means that the policy is evaluated each time the core process is executed. In this case, any applications ("app") that are being monitored by Dynatrace are checked.

5.2.3.5 CPU Usage

In this scenario, the focus is on efficient management of CPU usage to ensure it stays within an acceptable range. The system monitors the CPU percentage of the application and takes actions based on the observed usage. The administrator just needs to define a threshold value for CPU usage along with the corresponding event-condition-action rule. Then the system checks if the current CPU usage exceeds that threshold. If it does, the system takes proactive measures by temporarily pausing the execution of CPU-intensive tasks. This pause allows the system to alleviate the CPU load and prevent potential performance issues.

```
CPU Usage:
On True
      If cpuUsage (host, app)>80
      Then Pause (app)
```

"`On True`" signifies that the process responsible for checking policy violations should periodically assess the condition and, when necessary, temporarily suspend the task to prevent CPU overuse.

5.2.3.6 Memory Usage

Memory usage management is a crucial aspect of system performance optimization. Similar to CPU usage management, the system strives to maintain memory usage within an acceptable range. By defining a high threshold value for memory usage and implementing an appropriate policy, the system ensures efficient memory utilization. The process begins with monitoring of the application's memory consumption. It should be noted that the measurement unit for memory usage is in bytes. If the observed memory usage exceeds the defined threshold, the

system takes proactive measures to regulate it. This may involve temporarily pausing the execution of memory-intensive tasks to alleviate the memory load. Additionally, if necessary, the system can handle memory assignment to address the excessive memory usage scenario. It is important to note that the system remains adaptive and responsive. When memory usage falls below the threshold, the paused tasks can be resumed to ensure smooth execution and optimal resource allocation.

```
Memory Usage:
On True
      If memoryusage (host, app) > 36000000
      Then assignMemory (app)
```

Similar to the previous metrics, "`On True`" implies that the condition is assessed each time with regard to the memory usage of the specified application on the host.

5.2.3.7 Alarms

Sometimes a city faces a disaster or a problem that needs immediate attention and this can be realized from the telemetry data pattern. For instance, if the room temperature rises abovethe normal threshold, the risk of fire can be inferred or if the wave length on the beach is higher than normal, it can be life threatening for the people who live near the beach. Sometimes sensors can be placed in locations that are very sensitive, like in a data centre where the temperature should be monitored regularly and in case it is near the threshold the cooling system should cool down the room as fast as possible. To deal with these scenarios we need to define some policies and determine the actions that need to take place accordingly. First the corresponding alarm should be generated and then the smart city management system should choose the proper action to be taken place. Different scenarios are explained in the following and the management algorithms are presented.

Temperature:

To manage temperature, there are two methods that can be employed. The first method involves defining alarms directly in the ThingsBoard platform and sending those alarms to the ASCMS. The second method involves retrieving temperature values from ThingsBoard and

implementing a policy based on predefined thresholds. In this method, a high-value threshold is established to determine when action needs to be taken. The policy can be defined such that if the temperature exceeds the threshold, a flag value is set to “True”, indicating the need to activate the cooling system. On the other hand, if the temperature is within the normal range, the cooling system remains off.

This policy provides a flexible approach to manage the temperature. By defining the threshold appropriately, it is possible to respond to temperature variations and ensure optimal conditions. The use of predefined policies allows for automated decision-making based on the received temperature data and also enables general policies to be defined as well as being able to use temperature data in combination with other conditions for creating more complex policies. By implementing this policy, the system can effectively control the cooling system based on the temperature conditions. This not only helps maintain a comfortable environment but also enables energy efficiency by activating the cooling system only when necessary.

Temperature:

```
On temperature_high(location,s)
      If temperature(s)>20
      Then coolingSystem (location(s), True)
```

Wave height:

To effectively monitor and manage wave height, we employ a multi-step process. Firstly, we retrieve wave height data from ThingsBoard. Based on our dataset, we establish that wave heights typically range from 0 to 1.5. To effectively implement our policy, we designate the upper limit as 1.

If the water level exceeds this predetermined threshold, we activate a flag, marking it as True. This flag serves as a trigger for an application that promptly notifies individuals living in close proximity to the water body. This notification system ensures that people are promptly informed of any potentially hazardous wave conditions. Additionally, we take into consideration the presence of a breakwater, which is a structure designed to reduce the impact of waves and provide protection. By setting the breakwater flag to True, we activate its

functionality, adding an extra layer of defense against wave energy. This breakwater serves as a barrier, dissipating or redirecting the force of incoming waves and creating a calmer zone of water behind it.

By implementing this comprehensive approach, our policy effectively handles wave height. We utilize accurate data retrieval, real-time notifications, and the activation of a breakwater when necessary. This ensures the safety and well-being of individuals living near the water, while also safeguarding coastal infrastructure and ecosystems from the potentially damaging effects of high waves.

Wave Height:

```
On True
      If waveHeight(s)>1
      Then breakWater(beach_name, True) and
      notifyCitizens(beach_name)
```

Water quality:

Our primary goal is to guarantee the meticulous monitoring of water quality for a specific beach and implement effective management strategies based on this assessment. In pursuit of this objective, we place significant emphasis on the evaluation of turbidity, a pivotal parameter within our dataset that serves as an indicator of water quality. Turbidity refers to the measure of relative clarity or cloudiness of water, indicating the presence of suspended particles and impurities.

Within our dataset, turbidity values span a specific range, typically varying from 0 to 1683.48. For optimal water quality, it is desirable to maintain turbidity below a certain threshold. Lower levels of turbidity signify clearer water and better conditions. Based on recommended guidelines, an ideal turbidity value for recreational purposes would be below 1 nephelometric turbidity unit (NTU).

As part of our management policy, we establish the upper limit of turbidity at 50 NTU. This threshold serves as a crucial guideline, prompting us to take immediate action if the turbidity

exceeds this value. In such cases, it is of utmost importance to promptly notify individuals that swimming in the water is prohibited due to compromised water quality. In the event of high turbidity levels, which can pose potential health risks, the ASCMS triggers the activation of the notification system to ensure the safety and well-being of beachgoers by preventing them from swimming in water with high turbidity levels.

By incorporating turbidity monitoring and aligning it with our management policy, we prioritize water quality and strive to maintain a safe and enjoyable experience for visitors to the beach.

Water Quality:

```
On True
      If turbidity(s,beach_name)>50
      Then notifySwimmers(beach_name)
```

Traffic speed:

The ASCMS is also capable of tracking traffic speed and initiating relevant notifications when a slowdown is detected. This proactive feature serves to notify drivers in the vicinity to reduce their speed as they approach the affected area. The ASCMS can activate the notification system to interact with applications such as Google Maps to provide real-time updates, while dynamically adjusting traffic boards like Emergency Detour Route signage (EDR) to alert approaching drivers.

Furthermore, envisioning a future with advanced smart cities, where fully autonomous vehicles (level 5) are prevalent, our management system can extend its capabilities to communicate with these autonomous cars. By notifying the vehicles to reduce their speed, we can effectively prevent potential accidents from occurring. This integration of advanced technology ensures enhanced safety measures and contributes to the prevention of accidents in an increasingly connected and autonomous transportation landscape.

Traffic Speed:

```
On True
      If trafficSpeed(location) < 20
      Then notifyAutonomousVehicles(True, location) and
      notifyGoogle(True, location) and trafficSign(True,
      location)
```

5.2.3.8 Prioritization (Mission Critical vs Normal Data)

In a smart city environment, telemetry data can be classified into two categories: normal data and mission-critical data. Normal data originates from sensors such as parking sensors, and while important, it holds a lower priority compared to the data generated by sensors directly related to citizen safety, security, or critical tasks. Ensuring the utmost safety and well-being of citizens is paramount, which is why our smart city management system places special emphasis on processing mission-critical data efficiently.

To achieve this, our system allocates additional resources and prioritizes the transmission of mission-critical data. The key to this management task lies in determining the top priority among sensors or data streams. In our system, we accomplish this by assigning a priority attribute to each sensor, with values ranging from 1 to 9. Sensors with a priority value exceeding 5 are categorized as a high priority, resulting in an increase in data transmission from those sensors.

This policy allows us to dynamically adjust priorities based on the evolving needs of the smart city. As the priority of a sensor can change from normal to critical, we adapt our resource allocation and data transmission accordingly. By implementing this approach, we effectively manage mission-critical data, ensuring swift processing and response, thus enhancing the overall safety and efficiency of our smart city ecosystem.

Priority:

```
On True
      If priority(s)>5
      Then increaseTransmission(s)
```

In evaluating this policy, the policy evaluation process checks the status of sensors registered with ThingsBoard. If the priority of a sensor has increased beyond the threshold (5) then the policy is triggered.

5.2.3.9 Power Supply

Effective management of sensor power supply can be achieved through well-defined policies. While some sensors rely on batteries, operate on DC power, or even generate power internally, it is essential to implement strategies for managing sensor power supply in smart city systems.

To facilitate this management, the first step involves integrating circuits[2] into the sensors, enabling seamless switching between battery and DC power sources. Subsequently, the following policies can be established:

1. When the battery level falls below a predetermined threshold and the sensor is powered by a battery, the system automatically switches the power source to DC.

2. Conversely, if the sensor is powered by DC and the battery level reaches full capacity, the system reverts to using the battery as the power source.

3. In addition to batteries and DC power, solar energy can serve as an alternative power source. If solar energy becomes insufficient, the sensor can transition back to operating on battery power.

By implementing these policies, we ensure efficient power management, specifically addressing scenarios where the battery power becomes depleted. The ability to seamlessly switch power sources mitigates the risk of power outage and sustains uninterrupted sensor functionality.

---

[2] https://www.engineersgarage.com/automatic-power-supply-switching-for-battery-operated-devices-part-8-9/

The Power Supply policy comprises two distinct rules, one for situations when the battery level is low and another for when it reaches full capacity. These rules, denoted as "`battery_low`" and "`battery_high`," are encapsulated within a singular policy named "***`Power Supply`***".

`On battery_low`: The ASCMS monitors sensors to determine if they are operating on battery power. When the battery level falls below the predefined low threshold, the system triggers the `SwitchToDC(s)` event. This event promptly notifies the IoT Platform to transition the sensor from battery to DC power. This streamlined and automated approach optimizes power management, ensuring efficient resource utilization within the smart city ecosystem.

```
Power Supply:
On battery_low(s)
      If batteryLevel(s)<20 and powerSource == “battery”
      Then switchToDC(s)
```

`On battery_high`: The ASCMS monitors sensors that are not powered by their own battery and checks if their battery level has reached a specific threshold. If the battery level is at 100% and the power source is currently set to `DC`, the system triggers the `switchToBattery(s)` action. This action initiates a seamless transition of the sensor’s power source back to the battery.

```
On battery_high(s)
      If batteryLevel(s)==100 and powerSource == "DC"
      Then switchToBattery(s)
```

5.2.3.10 Calculation Based Policies

In addition to general policies, specific guidelines can be established to determine appropriate actions when the result of a calculation matches a predefined value. To achieve this, the IoT platform computes the outcome of a formula based on telemetry data received from sensors. The resulting answer is then forwarded to the smart city management system, which employs it to make informed decisions regarding necessary actions. Furthermore, the ASCMS is capable of independently calculating this information.

For instance, in a parking scenario, if the number of vehicles in the parking lot equals the total number of available parking spots, the Autonomic Smart City Management System (ASCMS) promptly signals the corresponding application to close the entrance gate. Simultaneously, it triggers the red coloration of the empty spot indicator or displays a message indicating that the parking lot is full. Furthermore, the ASCMS can calculate the number of available spots by subtracting the occupied spots from the total parking capacity and display this information on a garage status monitor. Additionally, this data can be transferred to a mobile application accessible to drivers, enabling them to stay informed about the availability of parking spots.

Parking:

```
On parkingFull(parking)
      If vehicleCount(parking)== totalSpaces(parking)
      Then updateParkingStatus(parking)
```

Table 1: Defined Policies

Domain	Policy	Definition
Sensor attributes management	Sensor firmware version	***Sensor firmware version:*** On NewFirmwareVersion(s) If firmwareVersion(s)! = latestVersion(s) Then UpdateSensorVersion(s)
	Sensor offline	***Sensor Offline:*** 1.On SensorOffline(s) If lastConnectTime(s)<currentTime Then Reboot(s) 2.On SensorOffline(s) If lastConnectTime(s)<currentTime and reboot(s)==True and HasDC(s) Then SwitchToDC(s) 3.On Sensor Offline(s) If lastConnectTime(s)<currentTime and reboot(s)==True and powerSource(s)= "DC" Then NotifyOperator ()

	Sensor priority	***Priority:*** On True If priority(s)>5 Then increaseTransmission(s)
	Power supply	***Power Supply:*** 1.On battery_low(s) If batteryLevel(s)<20 and powerSource == "battery" Then switchToDC(s) 2.On battery_high(s) If batteryLevel(s)==100 and powerSource == "DC" Then switchToBattery(s)
Infrastructure performance management	Response Time	***Check_Response_Time:*** On True If responseTime(app)> 10000 Then addComputeNodes (app)
	CPU Usage	***CPU Usage:*** On True If cpuUsage (host, app)>80 Then Pause (app)
	Memory usage	***Memory Usage:*** On True If memoryusage (host, app) > 36000000 Then assignMemory (app)
Sensor measurement and environment management	Temperature	***Temperature:*** On temperature_high(location,s) If temperature(s)>20 Then coolingSystem (location(s), True)
	Wave height	***Wave Height:*** On True If waveHeight(s)>1 Then breakWater(beach_name, True) and notifyCitizens(beach_name)
	Water quality	***Water Quality:*** On True If turbidity(s,beach_name)>50 Then notifySwimmers(beach_name)

	Traffic speed	***Traffic Speed:***
		`On True`
		`If trafficSpeed(location) < 20`
		`Then notifyAutonomousVehicles(True, location) and notifyGoogle(True, location) and trafficSign(True, location)`

	Parking	***Parking:***
		`On parkingFull(parking)`
		`If vehicleCount(parking)== totalSpaces(parking)`
		`Then updateParkingStatus(parking)`

In the next Chapter, we delve into the practical implementation of the policies discussed earlier, aiming to manage various facets of a smart city autonomously, without human intervention. Through a series of carefully designed experiments, we evaluate the effectiveness of our approach by comparing the state of the city before and after the deployment of our Autonomic Smart City Management System (ASCMS).

We analyze the outcomes to showcase the tangible improvements achieved with the ASCMS in place, highlighting the contrast between the normal operating conditions and scenarios where issues arise or when the smart city deviates from the desired conditions. This comparison serves to illustrate how the ASCMS acts promptly and effectively to manage and rectify the smart city ecosystem, ranging from environmental factors to critical infrastructure components.

Chapter 6 Experiments and Evaluation

In this Chapter, we will present a number of different scenarios in which we illustrate our autonomic management system for our prototype smart city. We first begin by describing the experimental environment for our smart city including the number of simulated sensors and the datasets providing the data for those sensors, the attributes of the sensors and the infrastructure of the simulated smart city. We will also describe the metrics and measurements that are collected (or can be collected) by our autonomic management system. This is followed by several sections that present different scenarios and experiments illustrating the operation of our autonomic management system.

6.1 Experimental Configuration

We first introduce our smart city environment and then describe the metrics and measurements used by the autonomic management system.

6.1.1 Smart City Configuration

Our prototype smart city consists of 41 sensors and devices and 4 network nodes, 1 computational node. The network configuration is illustrated in Fig. 6.1.

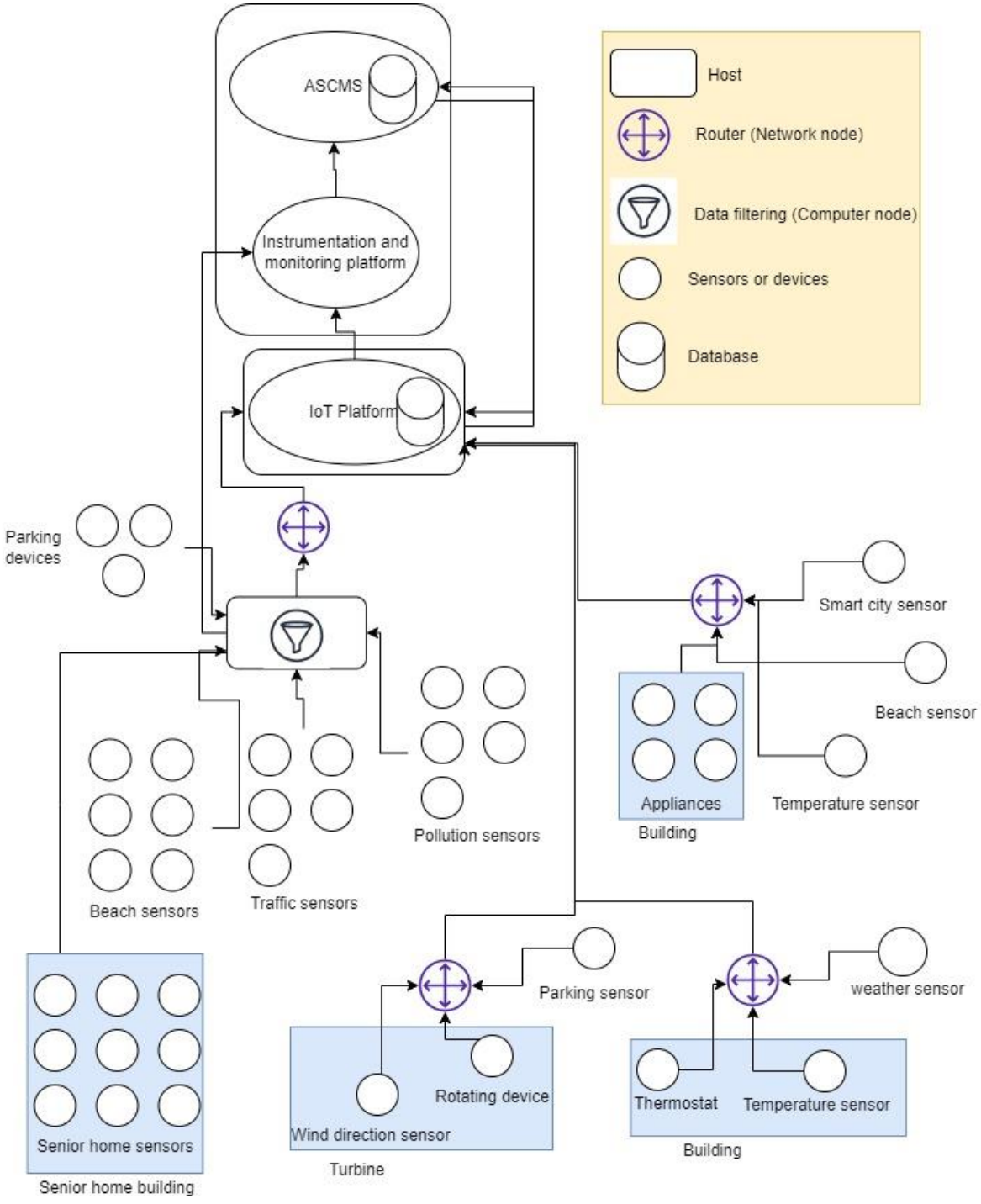


Figure 6.1 Experimental Configuration

The system configuration hosting both the smart city prototype and ASCMS prototype includes:

- Processor: Intel(R) Core (TM) i5-7500T CPU @ 2.70GHz with a processing speed of 2.71 GHz
- Memory: 8GB RAM
- Operating System: Windows 10; 64-bit, 64-bit operating system

Data for the sensors in our prototype smart city comes from the following associated datasets:

There are 6 sensors that provide data on beaches and water. There is one dataset for each of the sensors with the following information:

- Beach Name: name of beach where measurement takes place.
- Timestamp: The date and time when the measurements were taken.
- Water Temperature: water temperature in Celsius degrees.
- Turbidity: Water turbidity in Nephelometric Turbidity Units (NTU). As mentioned before "Turbidity" is the cloudiness or haziness of a liquid caused by suspended particles.
- Transducer Depth: Transducer depth in meters.
- Wave Height: Wave height in meters.
- Wave Period: Wave period in seconds.
- Measurement ID: A unique record ID made up of the Beach Name and Measurement Timestamp.
- breakWater: A flag representing if the breakwater is activated or not (added extra).

The remaining sensors all make use of different data sets that are from the city of Aarhus, Denmark generated for August 2014 which includes:

Five sensors that provide traffic data:

- AvgSpeed: average measured speed
- Timestamp: Date and time
- vehicleCount: number of vehicles

Five sensors that provide the pollution data and air quality:

- Ozone.
- Particullate_matter.
- Carbon_monoxide.
- Sulfure_dioxide.
- Nitrogen_dioxide.
- Timestamp.

Three parking devices that provide data about parking garages:

- Vehiclecount – number of vehicles in garage at reporting time.
- Updatetime – date and time at which data is reported.
- Totalspaces – max spaces in garage.
- Garagecode – code name of garage.

One sensor with weather related data:

- Date.
- Time.
- Temperature.
- Humidity.
- Dewpoint.
- Air Pressure.
- Wind Direction.
- Wind Speed.
- CoolingSystem.

and 21 devices and sensors that are added to evaluate the scalability and adaptability of the ASCMS including:

- 4 test devices that are assigned to a special customer.
- 9 sensors that record the information for senior homes.
- 1 sensor that only provides the temperature of a room.
- 1 wind direction sensor
- 1 rotating device that rotates based on the wind direction.
- 1 thermostat sensor and 1 thermostat that can control the temperature.
- 1 smart city sensor that is added to record all the data in one place.
- 1 beach sensor that is another test sensor and can provide some information such as temperature, water quality and wave height.
- 1 parking sensor that is for test.

All sensors in our smart city prototype have the following attributes:

- Version.
- Priority.
- LastConnectTime.
- BatteryLevel.
- PowerSource.
- Reboot: a flag that represents whether the device or sensor rebooted or not.

Incorporating all these sensors into the IoT platform enables us to demonstrate the utility of our model and showcase the functionality of our ASCMS.

6.1.2 ASCMS Configuration

In addition to getting data about the attributes of the sensors and sensor measurements, the ASCMS has the capacity to receive a multitude of performance metrics from the monitoring platform. Dynatrace can provide network traffic, connectivity, retransmission, throughput, latency, available disk percentage, and so on. However, our primary concentration lies in the management of select metrics, including response time, host CPU usage, and host memory utilization.

The rest of this chapter presents various scenarios executed in this simulated smart city environment to demonstrate the system's capabilities. To illustrate the operation and behavior of our ASCMS, we present screenshots of the IoT platform both before, during and after running the system. During the management system's operation, we capture and provide screenshots of the outputs, demonstrating the corrective actions in progress. For each scenario, we will also provide a chart illustrating the policy violations over time.

6.2 Scenario 1: Management of Sensor Attributes

In this scenario, we illustrate the effective management of sensors' attributes. We conduct various experiments to monitor and control crucial aspects such as sensor firmware versions, sensor priority, sensor offline and online status, and sensor power supply. The aim is to demonstrate the management of these sensor attributes within a smart city environment, as well as for multiple sensors collectively.

Our investigation also encompasses the dynamic alteration of these attributes while the autonomic management system is actively functioning. By doing so, we illustrate the system's capability to handle attribute changes in real-time, thus ensuring its efficiency and adaptability in a rapidly evolving environment.

Managing sensor attributes is a vital component in optimizing the performance and functionality of sensor networks. By effectively controlling the firmware version, we can ensure that all sensors in a smart city are equipped with the latest updates and enhancements,

guaranteeing their compatibility with other components within the system. Additionally, the ability to adjust the sensor priority by administrators enables the ASCMS to allocate resources appropriately or change the data transmission accordingly, ensuring that critical tasks are prioritized and addressed promptly. The priority assigned to sensors can be dynamic and can vary based on the prevailing situation. In certain circumstances, an administrator may give a particular sensor a higher priority for a specific period and then sometime later its priority would revert to normal by setting a policy or by doing it manually. Additionally, in the event of a critical occurrence or disaster, if a relevant policy is defined, the ASCMS can automatically adjust sensor priorities without requiring manual intervention. For instance, if the Wave Height surpasses the dangerous threshold, ASCMS can swiftly assign the highest priority to sensors near the beach to expedite data processing and activate protective measures promptly. Conversely, there may be instances when a sensor's priority returns to a normal level. This could occur when the urgency of a specific situation subsides, or when other sensors become more critical due to evolving conditions. Adapting the priorities accordingly ensures that resources are distributed appropriately, optimizing the overall performance and responsiveness of the sensor network.

Monitoring the offline and online status of sensors is crucial for maintaining the overall system's reliability. By actively managing this attribute, the system can promptly identify and rectify any connectivity issues, minimizing potential disruptions in data collection or system operations. Furthermore, managing the sensor power supply plays a pivotal role in ensuring uninterrupted functionality, preventing power-related failures and optimizing energy consumption.

The experiments conducted in this study aim to illustrate the effectiveness and efficiency of our approach to management within a dynamic smart city context. By observing the autonomic management system's response to attribute changes in real-time, we can demonstrate its ability to adapt and adjust seamlessly, ensuring the optimal performance and reliability of the entire sensor network.

6.2.1 Relevant Policies

The Autonomic Smart City Management System (ASCMS) operates based on the policies established by the system administrator. These policies serve as guidelines for managing the various sensor attributes, which are outlined as follows.

6.2.1.1 Sensor Version

To ensure optimal functionality and compatibility, it is important that the sensors get updated when a new version of the sensor firmware is released. We assume that it is the responsibility of the administrator to track versions of sensor firmware, e.g., through notices from vendors. Once the latest version has been identified by the administrator, we assume that it is stored in a database or a reference to it is stored in the data. The administrator can define policies that enable the ASCMS to check versions. If a disparity exists between the version numbers of latest version and the current version, the system initiates the necessary steps to update the sensor's firmware version to the most recent one. The firmware version can also be obtained from the manufacturer, provided they offer the appropriate API for this purpose.

The ASCMS automates the process of updating sensor firmware versions, alleviating the burden on administrators and eliminating the need for manual intervention to update firmware for every sensor. By streamlining this aspect of attribute management, system administrators can focus their attention on other critical tasks and defining policies while maintaining an up-to-date and optimized sensor network.

The policy for updating the sensor version is described below:

```
Sensor Version:
On NewFirmwareVersion(s)
      If firmwareVersion(s)! = latestVersion(s)
      Then UpdateSensorVersion(s)
```

The `NewFirmwareVersion(s)` indicates that when a new version is released for the specific sensor, and the sensor `firmwareVersion(s)` diverges from the `latestVersion(s)` designated by the administrator in the knowledge base, the system takes action to promptly update the sensor version to align with the latest version.

6.2.1.2 Sensor Priority

After defining the relevant policy, the ASCMS can manage and increase sensor data transmission, when it identifies a high sensor priority. This capability allows for data handling and prioritization based on the defined sensor priority threshold.

This policy proves to be particularly valuable in scenarios where the priority of data may fluctuate within a smart city environment. A prime example is during times of disaster or emergency situations, where certain areas of the city require immediate attention and prioritization. In such cases, sensors deployed in the affected region can be assigned higher priority levels, ensuring that their data is transmitted rapidly and given precedence over data from other sensors.

Below is the policy that helps with managing the sensor priority:

```
Priority:
On True
      If priority(s)>5 (5 is the threshold)
      Then increaseTransmission(s)
```

Within this policy, the initial step entails the administrator establishing a defined `threshold`. Subsequently, the condition assesses whether the priority of the sensor "`s`" exceeds the set threshold. If this condition is met, the policy triggers the `increaseDataTransmission(s)` function, thereby elevating the data transmission rate of the particular sensor. This increase can happen by adding more resources or increasing bandwidth.

6.2.1.3 Sensor Offline

In certain situations, a sensor may experience an offline status due to various factors, such as low battery, software issues, or hardware malfunctions. To address this problem, several potential solutions can be considered. Firstly, if the sensor encounters a software issue, a simple reboot may suffice to resolve the problem. However, if the issue persists, it could indicate that the battery has drained, necessitating the possibility of connecting the sensor to a direct current (DC) power source for recharging if that option exists for the sensor. However,

achieving this capability might not be feasible for all sensors, particularly those utilizing disposable batteries, like AAA batteries. For rechargeable batteries, a circuit could be integrated to facilitate the transition from battery power to DC power.

In the event that neither rebooting nor connecting to a DC power source resolves the problem, it is likely indicative of a hardware or major issue. In such cases, it becomes imperative for the system to promptly notify the administrator about the problem to initiate appropriate troubleshooting measures.

To facilitate the implementation of these solutions, a policy has been devised based on the relationship between the sensor's last connection time and the current time. The `lastConnectTime` is initially recorded as a Unix timestamp and needs to be converted to a DateTime format for comparison with the current time. The policy is outlined as follows:

- If the sensor goes offline and the `lastConnectTime` is earlier than the `currentTime`, initiate a reboot to attempt to resolve software issues.

  Sensor_Offline:
  ```
  On sensor_offline(s)
        If lastConnectTime(s)<currentTime
        Then reboot(s)
  ```

- If the sensor is still offline, the `lastConnectTime` is earlier than the current time, and a reboot was successfully performed (reboot = true), connect the sensor to a DC power source to address potential battery depletion.

  Sensor_Offline_Reboot:
  ```
  On SensorOffline(s)
        If lastConnectTime(s)<currentTime and reboot(s)==True and
        HasDC(s)
        Then SwitchToDC(s)
  ```

- If the sensor is still offline, the `lastConnectTime` is earlier than the current time, a reboot was successfully performed, the `power_source` is DC, and the problem persists, notify the operator or administrator of the situation, indicating a potential hardware or major issue.

Sensor_Offline_PowerSource:
```
On sensor_offline(s)
      If lastConnectTime(s)<currentTime and reboot(s)==True
         and power_source(s)==" DC"
      Then Notifyoperator()
```

This combination of policies helps the system ensure a systematic and prioritized approach to addressing sensor offline situations. It combines logical steps such as rebooting, and power source switching with timely notifications to facilitate efficient problem resolution and minimize disruptions in data collection and system operations.

6.2.1.4 Sensor Power Supply

The power supply of the sensor is managed based on the level of its battery. The system incorporates a set of rules to ensure efficient power management. If the power supply is currently running on the battery and the battery level drops below a specified threshold, the system initiates a switch to the direct current (DC) power source. This switch helps prevent the sensor from losing power and ensures uninterrupted operation. Conversely, if the battery level reaches its maximum capacity and the sensor is connected to the DC power source, the system recognizes that the battery is fully charged. In this case, it is necessary to disconnect the sensor from the DC power source to avoid overcharging or wasting power.

To provide flexibility and customization, the threshold value is defined by the system administrator. Administrators can set a specific threshold that aligns with the unique requirements and capabilities of the sensor. This allows for adaptability to different battery capacities, power consumption patterns, and operational needs.

The power management policy can be summarized as follows:

- `On Battery_low`: If the current battery level falls below the threshold defined by the administrator (`battery_level(s)<threshold` "where s represents a specific sensor"), the system triggers a switch to the DC power source. This ensures a continuous power supply and prevents the sensor from running out of battery. The threshold is set to 20 in this case.

Sensor_Battery_Low:

```
On Battery_low(s)
      If battery_level(s)<threshold and powerSource == “battery”
      Then SwitchToDC(s)
```

- `On Battery_high`: If the battery level reaches 100% capacity and the sensor is connected to the DC power source, the system recognizes that the battery is fully charged. To optimize power usage, the sensor is then disconnected from the DC power source, allowing it to rely solely on its battery for power.

Sensor_Battery_Full:

```
On Battery_high(s)
      If batteryLevel(s)==100 and powerSource == "DC"
      Then switchToBattery(s)
```

By implementing this power management policy, the system ensures that the sensor always has a reliable power source while efficiently utilizing energy resources. This approach maximizes the sensor's operational uptime, prevents unnecessary power drain, and contributes to a sustainable and efficient power management strategy.

6.2.2 Managing Sensor Attributes

In our first experiment, we will demonstrate how the Autonomic Smart City Management System (ASCMS) manages sensor attributes in various scenarios. Within the ASCMS framework, a process is integrated to proactively monitor sensor attributes from the IoT Platform (ThingsBoard) and initiate corresponding policies when the specified conditions are met.

The ASCMS exhibits its adaptability by checking the sensor attributes and dynamically adjusting them in real time, swiftly responding to changes that occur while it is running. This capability allows the ASCMS to maintain optimal attribute configurations and promptly address any deviations from the expected conditions.

Fig. 6.2 captures a screenshot from the IoT platform. It shows the sensor attributes that can be adjusted manually for one of the smart city sensors, in this case, the sensor name is “Sensor 39”.

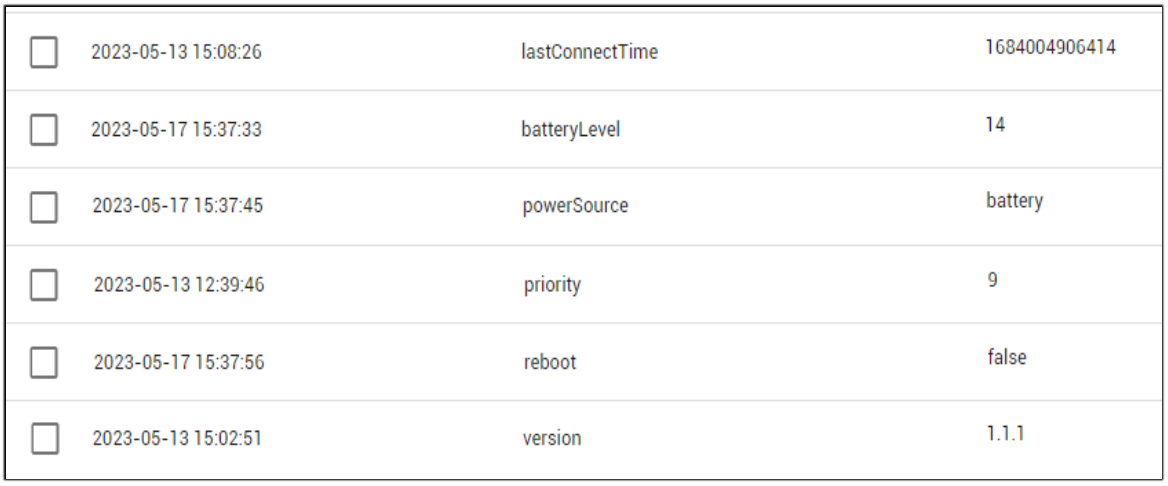


Figure 6.2 Sensor Attributes for “Sensor 39” before action.

In this scenario, we assume that the low threshold for the battery level of “Sensor 39” is 20. Based on the data about the sensor in the IoT platform, however, the batteryLevel is currently measured at 14, indicating a low charge. When the ASCMS identifies a policy that evaluates to TRUE, it carries out the actions specified in that policy. It recognizes the battery level falling below the defined threshold as shown in Fig. 6.3 and takes appropriate action to address the situation. In response, the ASCMS initiates a power source change, transitioning the sensor power source from its current state to the direct current (DC). As a result of this transition, the sensor's battery level begins to rise gradually, and the updates are getting published to the ThingsBoard (Fig. 6.3). As is apparent, the initial battery level commenced at 14. With each subsequent increase, the updated value is transmitted to ThingsBoard. In the provided screenshot, the reading stands at 30, yet the battery level steadily ascends until it reaches a full charge of 100. The ASCMS actively manages and monitors the attribute in real time using the autonomic feedback loop, ensuring that the battery level is efficiently going up. Screenshots in Fig. 6.3 to 6.5 depict the live output of the ASCMS during its runtime to trace what the system does while it is working step by step. It is essential to note that these individual figures appear sequentially in the output. For the sake of clarity and improved representation, we have divided them into several separate figures.

```
battery 14
Attributes successfully published to Thingsboard
15
Attributes successfully published to Thingsboard
16
Attributes successfully published to Thingsboard
17
Attributes successfully published to Thingsboard
18
Attributes successfully published to Thingsboard
19
Attributes successfully published to Thingsboard
20
Attributes successfully published to Thingsboard
21
Attributes successfully published to Thingsboard
22
Attributes successfully published to Thingsboard
23
Attributes successfully published to Thingsboard
24
Attributes successfully published to Thingsboard
25
Attributes successfully published to Thingsboard
26
Attributes successfully published to Thingsboard
27
Attributes successfully published to Thingsboard
28
Attributes successfully published to Thingsboard
29
Attributes successfully published to Thingsboard
30
```

Figure 6.3 Managing the Battery Level.

Other than the batteryLevel, the ASCMS identifies through policy ***Sensor_Offline*** that the lastConnectTime of the sensor differs from the current time (Fig. 6.4) indicating that the sensor is currently offline. To rectify this issue, the system takes action by rebooting the sensor. The reboot proves successful, resolving the offline status and restoring the sensor's functionality.

Moreover, during this process, the ASCMS recognizes that the sensor's priority level is 9, surpassing the threshold of 5 (Fig. 6.4) based on policy ***Priority***. In reaction, the system exhibits dynamic adaptability by fine-tuning the data transmission rate for enhanced speed. Consequently, the pace of data transmission experiences an upswing, manifesting as a noticeable surge in the number of outputs generated within a designated timeframe. Conversely, when the priority falls below a threshold of 5, data transmission adopts a more conservative pace, resulting in fewer outputs produced over the same duration. This prioritization ensures that the data from the sensor with a higher priority level is transmitted at a faster rate compared to other sensors, enabling timely and efficient data delivery.

```
sensorOffline
Attributes successfully published to Thingsboard
2023-05-17 15:59:41.973000 2023-05-17 16:00:01.954818
 The sensor is restarting ...
Action executed:  makeSensoronline
priority_high
9 5
High priority detected. Increasing data collection frequency...
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
Mission-critical data
```

Figure 6.4 Managing the sensor when it is offline and has high priority.

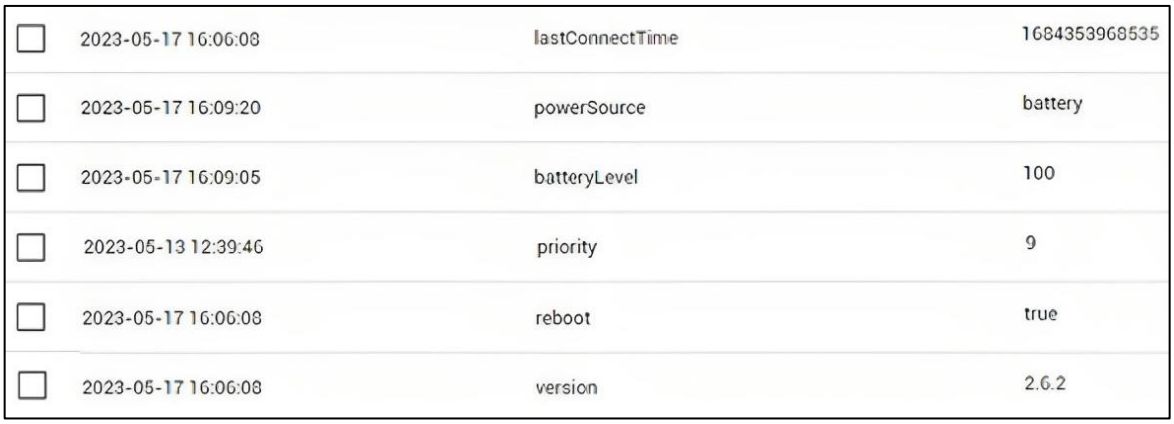
```
version_old
True
2.6.2
Attributes successfully published to Thingsboard
Action executed:  changeLatestversion
```

Figure 6.5 Managing sensor version.

Continuing the attribute analysis, the ASCMS proceeds to check the sensor's firmware version. It detects that the current version is 1.1.1, which differs from the latest available version of

2.6.2. Understanding the importance of maintaining up-to-date firmware, the system initiates a version update, changing the sensor's firmware to the latest version (Fig. 6.5).

The results of these attribute management actions are depicted in Fig. 6.6.



☐	2023-05-17 16:06:08	lastConnectTime	1684353968535
☐	2023-05-17 16:09:20	powerSource	battery
☐	2023-05-17 16:09:05	batteryLevel	100
☐	2023-05-13 12:39:46	priority	9
☐	2023-05-17 16:06:08	reboot	true
☐	2023-05-17 16:06:08	version	2.6.2

Figure 6.6 Sensor attributes after the ASCMS changes.

Notably, the lastConnectTime attribute reflects an update as a result of the successful reboot. Additionally, the version attribute demonstrates the firmware upgrade to the latest version, now registered as 2.6.2. Furthermore, the power source attribute reflects a change as well. Initially, the sensor was connected to the DC power source, but as the battery level reaches its maximum capacity, the system disconnects the sensor from the DC power source, switching it back to the battery power source.

This comprehensive attribute management performed by the ASCMS showcases its capability to efficiently handle various aspects of sensor operation and attributes. By addressing offline status, adjusting data transmission rates based on priority, updating firmware versions, and managing power sources, the ASCMS ensures the optimal functioning of sensors within the smart city ecosystem.

6.3 Scenario 2: Management of Smart City Infrastructure Using Performance Metrics

In this scenario, the goal is to manage the smart city infrastructure using performance metrics. Although there are so many performance metrics that can be received from Dynatrace, metrics that are chosen for the management include CPU usage, memory usage and response time. These metrics are extracted from the host housing the IoT platform. This section is specifically focused on controlling the smart city infrastructure using these factors with our Autonomic Smart City Management System (ASCMS).

6.3.1 Relevant Policies

In this part, we provide a summarized presentation of the policies established for smart city infrastructure management, leveraging performance metrics. This overview serves as a convenient point of reference.

6.3.1.1 CPU Usage

The policy for controlling CPU utilization aims to maintain utilization within defined limits. The ASCMS monitors the CPU usage of an application on the host and triggers action to manage performance. The administrator needs to specify a CPU usage threshold (here is set to 80 percent) and the corresponding event-condition-action rule. The system evaluates whether the current CPU usage surpasses this threshold and if it does, the system takes action by temporarily halting CPU-intensive tasks. This temporary pause aims to reduce CPU load and prevent any potential performance problems.

```
CPU Usage:
On True
      If cpuUsage (host, app)>80
      Then Pause (app)
```

6.3.1.2 Memory Usage

Another critical factor in optimizing the infrastructure performance is memory usage. The system's goal is to keep memory usage within acceptable bounds by setting a high threshold value for memory consumption and implementing a suitable policy for efficient memory

utilization. In this policy the administrator defines the threshold which is 36000000 and when the observed memory usage exceeds this threshold, the system intervenes to maintain control. This intervention might include temporarily suspending memory-intensive tasks to ease the memory load. Furthermore, if necessary, the system can redistribute memory allocations to address the situation of excessive memory usage.

```
Memory Usage:
On True
      If memoryusage(host, app)>36000000
      Then assignMemory (app)
```

6.3.1.3 Response Time

The policy to manage the response time is defined based on the fact that a response time of more than 10 milliseconds needs attention. Focusing on application response time, the policy employs vertical reactive scaling, adding resources to reduce response time. However, this approach is reactive and may not prevent overloads. To save resources and optimize performance, compute nodes can be reduced when the workload is light and response times remain under 10 milliseconds.

```
Check_Response_Time:
On True
      If responseTime(app)> 10000
      Then addComputeNodes (app)
```

6.3.2 Managing the Smart City Infrastructure

This scenario is developed to monitor and manage the smart city’s infrastructure and its performance with a specific focus on monitoring CPU usage, memory utilization, and response time. In this experiment, our primary attention centers on a dedicated host serving as the backbone for the IoT platform. The first performance metric under scrutiny is CPU usage. To comprehend the system's operation, we first delve into its workflow. Initially, the administrator sets a predefined threshold value, which, in this experiment, stands at 80. Subsequently, the system continually receives CPU usage data. It then evaluates a condition: if the CPU usage surpasses the set threshold of 80, the tasks are temporarily halted. The system

also displays the executed action, as depicted in Fig. 6.7. It is worth noting that after the execution of this action, the CPU usage experiences a decline. In this experiment the ASCMS prints CPU usage values to demonstrate that when it falls within the normal range, the ASCMS refrains from activating any policies since the specified condition remains unmet.

```
Current CPU usage: 77.0
high_cpuusage
Current CPU usage: 77.0
high_cpuusage
Current CPU usage: 71.3
high_cpuusage
Current CPU usage: 71.3
high_cpuusage
Current CPU usage: 78.8
high_cpuusage
Current CPU usage: 78.5
high_cpuusage
Current CPU usage: 84.7
high_cpuusage
CPU usage above threshold. Pausing task.
Action executed:  pause
Current CPU usage: 3.3
high_cpuusage
Current CPU usage: 71.3
high_cpuusage
Current CPU usage: 85.6
high_cpuusage
CPU usage above threshold. Pausing task.
```

Figure 6.7 Managing CPU Usage.

Another crucial performance metric to consider is response time, a key indicator of system efficiency and user satisfaction. In our pursuit of optimizing infrastructure performance, we have meticulously formulated a policy. This policy springs into action when the response time surpasses the critical threshold of 10. One of the primary actions taken is the dynamic allocation of additional compute nodes to the specific task at hand. By doing so, we proactively address performance bottlenecks, ensuring that the smart city infrastructure is optimized.

```
Response time: 15.987861633300781
high_responsetime
Compute node added
Response time: 0.28121113777160645
Action executed:  addComputenode
```

Figure 6.8 Managing Response Time while the ASCMS is working.

The last performance metric that we consider in this research is memory usage which is very important in maintaining infrastructure performance. The threshold for this metric is 36000000 bytes, meaning that if the memory usage goes above this value an action is executed and a simulated function is called that can print a message and also pause tasks and assign more memory to the host. The output of the system is provided in Fig. 6.9. The reason that the changed value of the memory usage is not displayed is that due to the complex memory management in Python processes, after executing an action, there is a delay before memory is released, preventing real-time updates to memory usage. Therefore, the reduction in reported memory usage may not be immediately visible, as memory is released gradually or as determined by the system's memory management strategies.

```
Current memory usage: 37875712
high_memoryusage
Memory usage above threshold. Pausing task. Assigning more memory
Action executed:  assignMemory
```

Figure 6.9 Managing Memory Usage while the ASCMS is working.

Fig. 6.10 illustrates the chronological sequence of policy violations for the management of smart city infrastructure. When a policy is breached, the system logs the incident (policy violation) along with its timestamp. This plot depicts the progression of policies executed by the ASCMS over time.

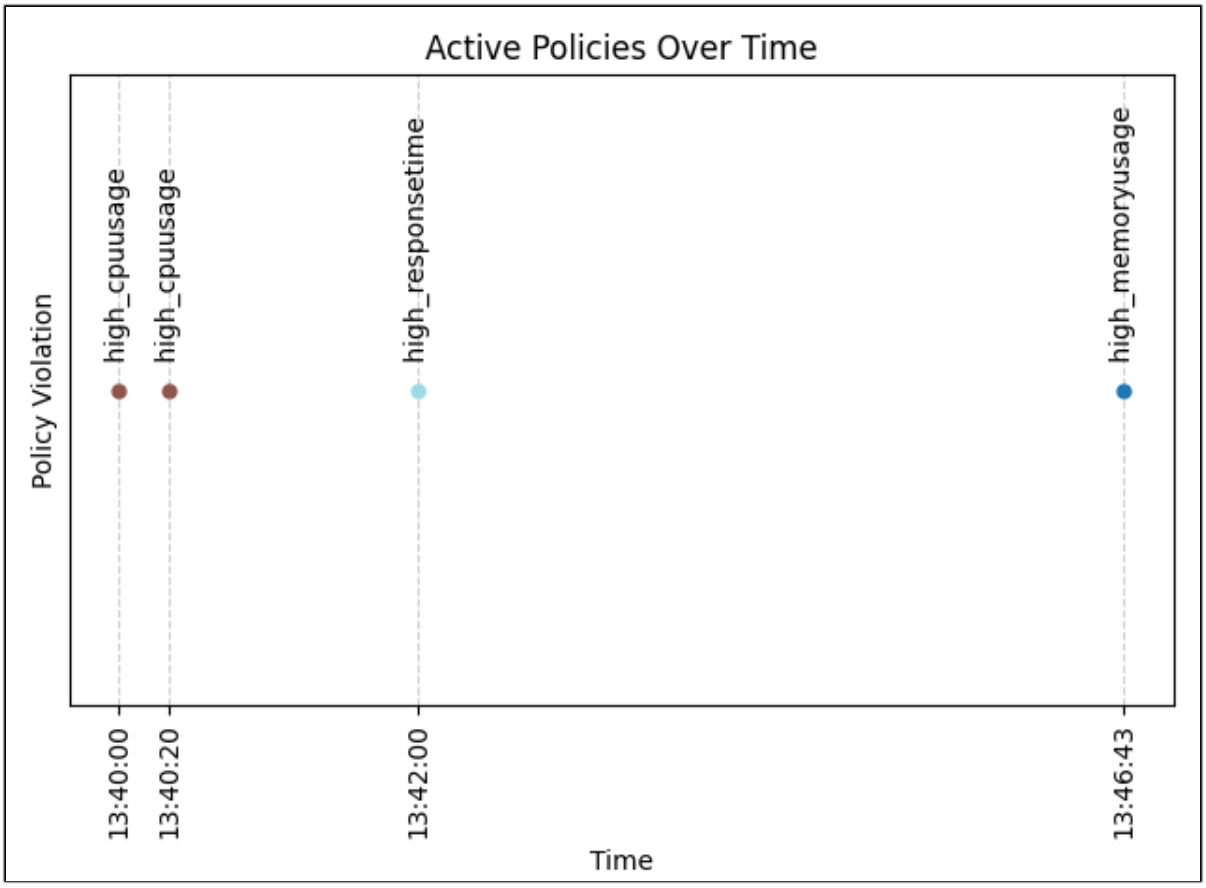


Figure 6.10 The sequence of policy violations over time.

6.4 Scenario 3: Management of Smart City Environment and Communicating with IoT Platform.

In this scenario, our focal point revolves around the management of the smart city environment, primarily facilitated through the collection of sensor telemetry data in real-time and communication with the IoT platform. For this scenario we have selected a range of metrics for monitoring and analysis such as temperature, wave height, water quality and parking status. We have to highlight that these metrics are coming from different specialized sensors. As is explained in section 6.1 there are several sensors such as temperature sensors, beach sensors, and parking devices. As highlighted in the previous scenario, it is important to recognize that these variables are dynamic and subject to change in real-time. This dynamicity serves as a testament to the ASCMS's prowess in effectively managing the smart city

environment under ever-evolving conditions. Sections 6.4.1.1 to 6.4.1.4 provide brief descriptions of the defined policies that are needed for this scenario.

6.4.1 Relevant Policies

In this section the policies for managing the smart city environment will be briefly presented. A detailed explanation of these policies can be found in Chapter 5, but we provide a concise summary for easy recall. These policies are defined to manage different environmental factors such as temperature, wave height, parking space, and water quality(turbidity). Events are determined by the policy execution process, which receives the telemetry data from the Perception Unit.

6.4.1.1 Temperature

The initial focal point of environment management lies in the realm of temperature. The temperature data is received from the temperature sensors embedded in the smart city and their data is received from the IoT platform. The process of changing the temperature can be managed by the IoT platform or a central thermostat but in this research, we take charge of overseeing and managing this metric to showcase the ability of our system to provide a high-level management of the smart city environment. In this policy, the administrator establishes a predefined threshold (20 degrees Celsius). We assume that the cooling system involved with this sensor is to be activated by the ASCMS.

Temperature:

```
On temperature_high(location,s)
      If temperature(s)>20
      Then coolingSystem (location(s), True)
```

6.4.1.2 Wave Height

In our discussion of datasets, one of our datasets pertains to telemetry data obtained from embedded sensors located at five distinct Chicago beaches. This dataset furnishes information regarding wave height, and we utilized its information as part of our hypothetical city for the assessment of our system. Notably, our analysis did not involve specific modelling for the city of Chicago. Additionally, we introduced an essential attribute known as the "breakwater,"

which bears the responsibility of fortifying coastal regions and maritime infrastructure. Its primary function is to mitigate the potentially damaging effects of waves, tides, and currents.

We have formulated a policy that hinges on the activation of the breakwater when the wave height exceeds a predefined threshold (1 meter). It is worth noting that while breakwaters are conventionally passive structures designed to offer protection against wave height, for the purposes of this study, we consider the hypothetical scenario where these structures can be activated to fulfill this vital role. Alongside the breakwater activation, we have incorporated an additional function designed to notify beachgoers in the vicinity, emphasizing the importance of being cautious about their own safety.

```
Wave Height:
On True
      If waveHeight(s)>1
      Then breakWater(beach_name, True)and
        notifyCitizens(beach_name)
```

6.4.1.3 Parking Space

In modern urban landscapes, every city is endowed with numerous parking spaces, each a crucial element of daily mobility. The efficient and autonomous management of these parking spaces has become paramount, minimizing the need for human intervention. In response to this need, we considered a policy to monitor and oversee the utilization of parking spaces within the smart city. In our dataset, we have three parking devices, equipped to detect and record the status of parking spots in real-time. Moreover, we possess accurate data regarding the total number of parking spots available in each designated area.

A policy has been defined such that when all parking spots in a given area reach full occupancy, it triggers an action. This action initiates a function that changes the parking sign to indicate that the parking area is at maximum capacity.

```
Parking:
On parkingFull(parking)
      If vehicleCount(parking)== totalSpaces(parking)
      Then updateParkingStatus(parking)
```

6.4.1.4 Water Quality

The sensors placed at beaches provide data about water turbidity levels. This particular variable serves as a valuable indicator for assessing water quality. We have defined a policy that takes measures when the turbidity levels in the water deviate from the ideal conditions for safe swimming. It activates a function designed to promptly alert swimmers to the prevailing water quality.

```
Water Quality:
On True
      If turbidity(s,beach_name)>50
      Then notifySwimmers(beach_name)
```

6.4.2 Managing the Smart City Environment.

Within this section, we systematically examine individual variables, though it is important to emphasize that the ASCMS can concurrently manage multiple variables. Our first variable of focus is temperature, as depicted in Fig.6.11. The initial temperature reading stands at 25 degrees Celsius, with the cooling system currently deactivated (set to 0). It is worth noting that the predefined threshold is set at 20 degrees Celsius. This implies that if the temperature exceeds this threshold, the cooling system should be activated, gradually reducing the temperature over time.

☐	2023-05-11 16:57:46	coolingSystem	0
☐	2023-05-11 16:57:46	temperature	25

Figure 6.11 Temperature measurement for “Temperature Sensor 1”.

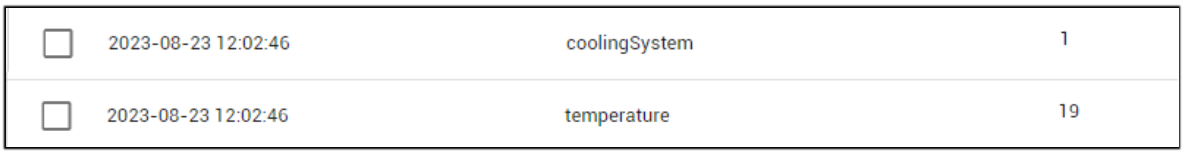
Fig. 6.12 illustrates the ASCMS in action. It initially receives a temperature reading of 25 degrees Celsius. Since this value surpasses the 20-degree threshold, the condition is met, and the system returns "True". Consequently, the prescribed action is executed, resulting in the activation of the cooling system. We have engineered a simulation function for the cooling system that accurately replicates its real-world functionality. Over time, the temperature

progressively declines, with each new data point being promptly transmitted to ThingsBoard for tracking and monitoring. This cycle persists until the condition is no longer met.

```
temperature_high
True
24
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
True
23
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
True
22
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
True
21
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
True
20
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
True
19
Attributes successfully published to Thingsboard
Action executed:  coolingSystem
temperature_high
Condition is not met
```

Figure 6.12 Managing Temperature while the ASCMS is working.

The outcome following the final action execution is illustrated in Fig. 6.13. As evident, the temperature has dropped to 19 degrees Celsius, falling below the 20-degree threshold, while the cooling system remains active.



☐	2023-08-23 12:02:46	coolingSystem	1
☐	2023-08-23 12:02:46	temperature	19

Figure 6.13 Temperature after action execution.

Fig. 6.14 illustrates the wave height data and also the breakwater's value which is added to control the wave height. Specifically, we focus on the Montrose Beach sensor (MBS). The predetermined wave height threshold is set at 1 meter. This threshold serves as a critical marker; when the wave height surpasses this value, an automated action is triggered.

To ensure the safety of the beachfront areas, we have simulated a function to activate the breakwater and simulate its functionality. This proactive measure guarantees the timely activation of the breakwater system, effectively shielding the adjacent beach areas from potential hazards. As evident in Fig. 6.14, the initial wave height value is 1.3, surpassing the predefined threshold. When this policy is violated, it triggers an action. Subsequently, the wave height gradually diminishes, as illustrated in Fig. 6.15, until it reaches a point where the condition is no longer breached (0.9). The values presented in Fig. 6.16 depict the state after the most recent action has been executed.

☐	2023-08-23 12:09:46	waveHeight	1.3
☐	2023-08-23 12:02:46	breakWater	0

Figure 6.14 Wave height information before the action.

```
waveHeight_high
True
1.2
Attributes successfully published to Thingsboard
Action executed:  breakWater
waveHeight_high
True
1.1
Attributes successfully published to Thingsboard
Action executed:  breakWater
waveHeight_high
True
1.0
Attributes successfully published to Thingsboard
Action executed:  breakWater
waveHeight_high
True
0.9
Attributes successfully published to Thingsboard
Action executed:  breakWater
waveHeight_high
Condition is not met
```

Figure 6.15 Managing wave height while ASCMS is working.

Figure 6.16 Wave height after action execution

In the realm of parking space management, the ASCMS continually compares the number of occupied parking spots to the total available spaces. When vacant spaces are detected, it initiates an action that triggers a function to display a message on the parking sign indicating that the parking space is not yet at full capacity. Conversely, when all parking spots are occupied, it invokes another function to alert people that the parking lot is indeed full.

Fig. 6.17 visually demonstrates the status of the parking area before action execution. If the number of vehicles in the parking lot remains lower than the total number of available spots, the corresponding output is as presented in Fig. 6.18. On the other hand, Fig. 6.19 illustrates a scenario where the total number of spots matches the vehicle count, signifying that the parking is at full occupancy. In this case, the output is determined by the information in Fig. 6.20 and the action executed is labeled as "UpdateParking".

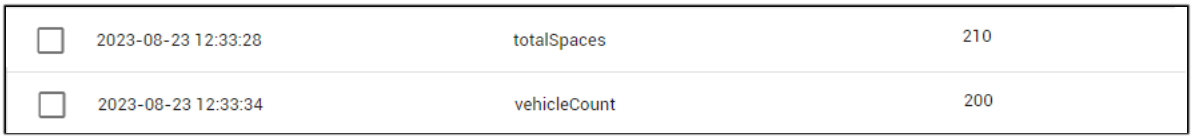


Figure 6.17 Parking information before the action.

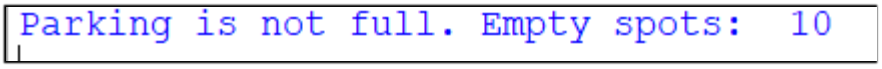


Figure 6.18 Parking sign when the parking lot is not full.

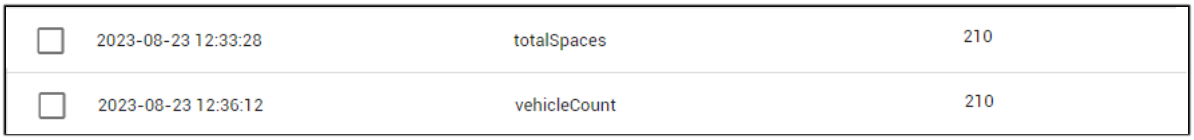


Figure 6.19 Parking information before the action.

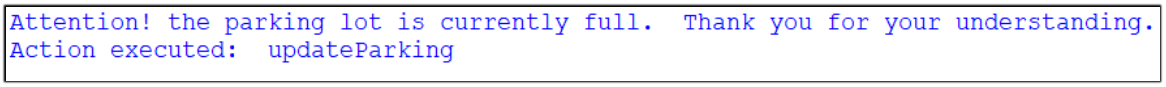


Figure 6.20 Parking sign when the parking lot is full.

Finally, we illustrate a policy dealing with water quality, with a particular focus on turbidity levels. As previously discussed, a turbidity reading exceeding 50 is indicative of unsuitable water quality for swimmers.

When turbidity is below this 50-point threshold, no immediate action is taken Fig. 6.21. However, when the turbidity level surpasses 50, as demonstrated in Fig. 6.22, reaching a value of 200, our predefined policy comes into effect. This policy activates a specific action: the

initiation of a function designed to generate an alert on the beach signage Fig. 6.23. This timely alert serves as a crucial safety measure, informing beachgoers of the current water quality conditions and helping to ensure their well-being.

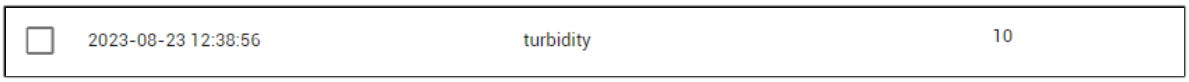


Figure 6.21 Turbidity information when the level is below the threshold.

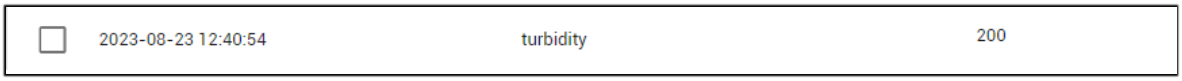


Figure 6.22 Turbidity information when the level is above the threshold.

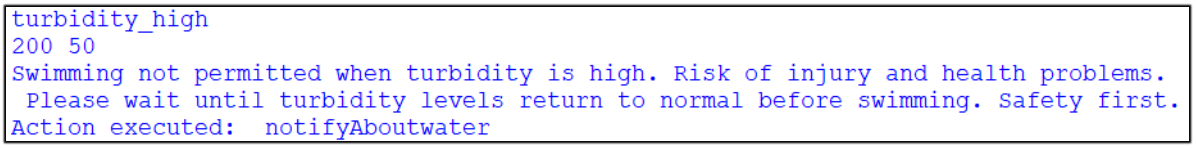


Figure 6.23 Alert when turbidity is above the threshold.

Fig. 6.24 visually represents the sequence of policy violation for smart city environment management.

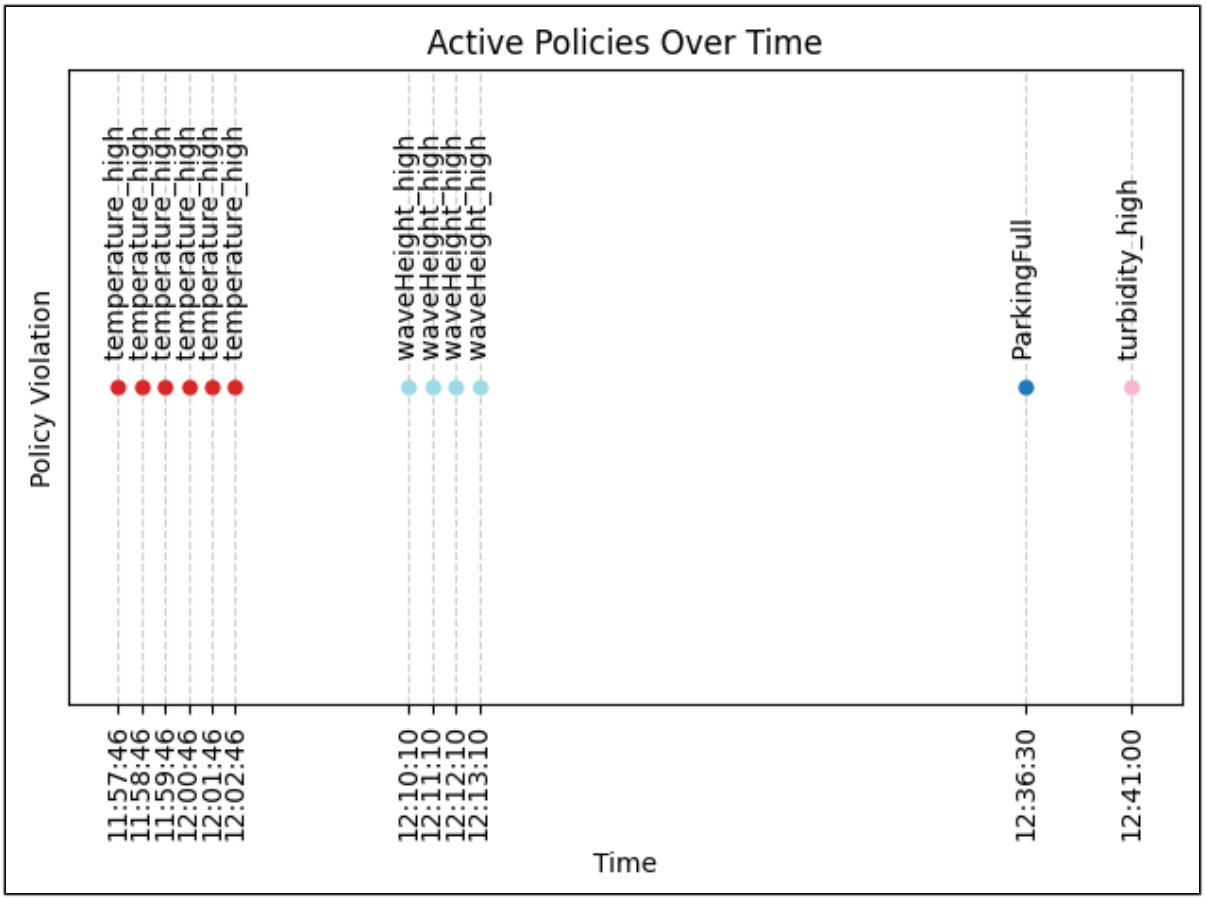


Figure 6.24 The sequence of policy violation over time for smart city environment management.

6.5 Scenario 4: Sensor Diversity and Policy Variability

In this scenario, we illustrate a complex amalgamation of challenges involving sensor measurements, attributes, and infrastructure performance that unfold at different times and in varying sequences to demonstrate the capabilities and adaptability of the Autonomic Smart City Management System (ASCMS). This scenario showcases how ASCMS can monitor and manage diverse factors under varying conditions and timelines.

These challenges include Sensor 1 operating with outdated firmware, an environment with elevated temperatures, a surge in CPU load, and Sensor 2 experiencing an offline status. These events occur at different times and in various orders.

In this scenario, the key focus lies on the chart representing the timeline of policy violations, as the output and before-and-after statuses remain consistent with previous scenarios. Fig. 6.25 illustrates the chronological sequence of policy breaches, offering valuable insights into how ASCMS identifies and responds to environmental change, deviations in sensor attributes and infrastructure performance issues.

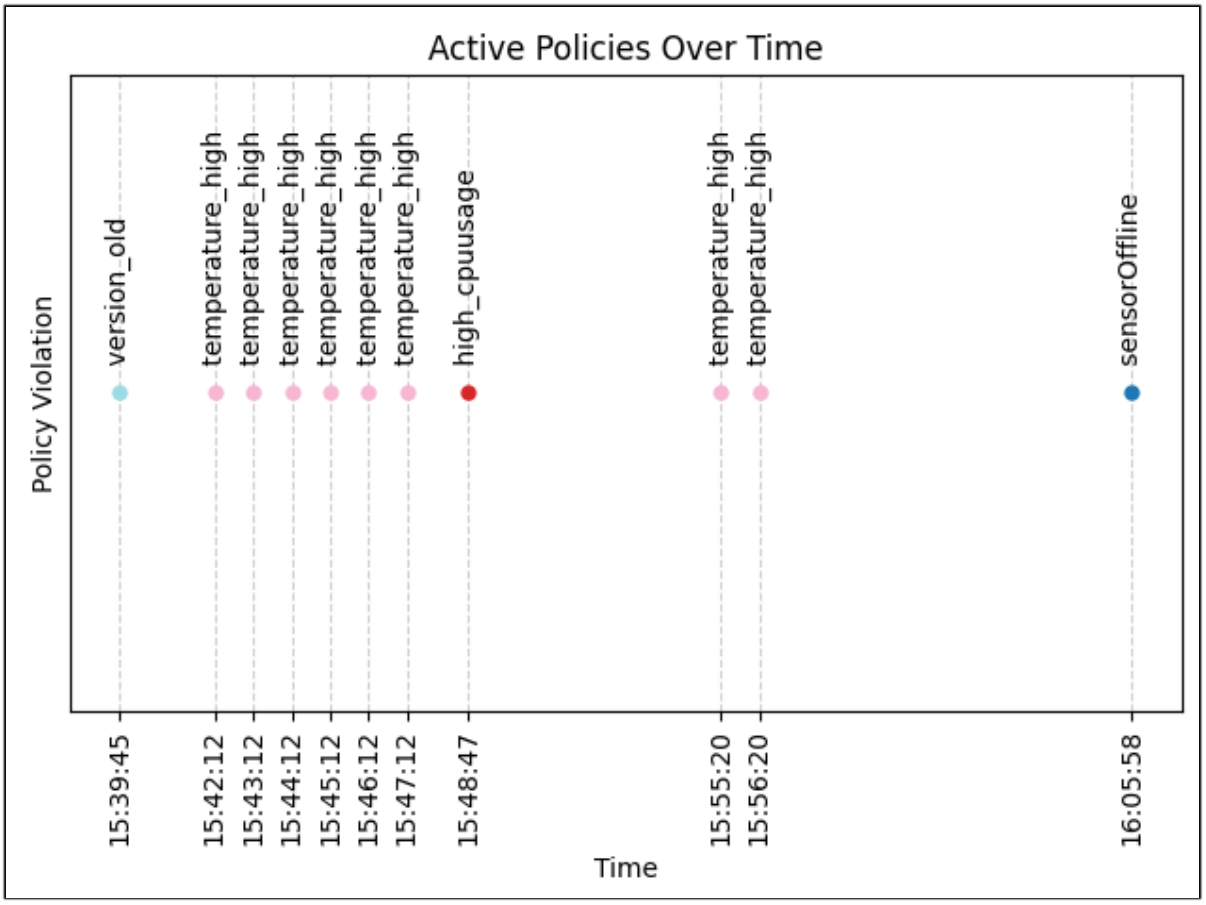


Figure 6.25 The sequence of policy violations over time for scenario 4.

6.6 Discoveries and Reflections: Insights from Experimental Implementation

Through the implementation of various scenarios and the execution of experiments, we gained valuable insights. Our initial observation was the feasibility of monitoring and managing infrastructural performance. Additionally, employing policies proved instrumental in reducing the need for human intervention; however, we recognized the importance of administrative

support in some areas, particularly in the face of hardware issues or actions requiring technical expertise. Another critical lesson emphasized the necessity for the control loop to iterate even for a singular metric or factor, continually assessing conditions. This adaptive approach proved vital, given the potential for abrupt changes in the smart city ecosystem. The experiments underscored the significance of leveraging an IoT platform for improved connectivity and integration with autonomous systems. Navigating diverse scenarios, including one with multiple policies, revealed the flexibility of our proposed system in simultaneously addressing various issues and policy violations. This experience underscored the importance of designing our solution to be versatile and adaptable, capable of handling a spectrum of real-world challenges. Furthermore, we recognized the efficacy of defining policies with multiple sub-rules, conditions, and actions, demonstrating the system's applicability to more complex environments.

Chapter 7 Summary, Conclusion and Future Work

Our research focuses on autonomic management methods to support the management of smart cities. Based on previous research, we have assumed that the smart city infrastructure has several layers: sensors and devices at the base tier to complex IoT platforms, third-party applications and managerial units at the upper tiers. Unlike previous work which has considered aspects of managing sensors and managing aspects of smart cities, we have considered the challenges in managing the end-to-end infrastructure of a smart city. Also, we have assumed that smart cities will utilize existing tools and applications, in particular relying on IoT platforms for handling sensors, applications for data filtering and monitoring and instrumentation systems for collecting the operational data.

Any smart city has numerous heterogeneous devices, applications, networks, hosts, etc. The management of the infrastructure will be daunting. We have therefore focused on the use of policy-based management as a means to automate and help manage the infrastructure. We have proposed an autonomic management system with the aim of managing our smart city. Using policy-based methods for this purpose makes autonomic management more straightforward by focusing on defining the policies and adjusting them.

In this research we addressed the following questions:

- How can we monitor the performance of the operational side of the smart city infrastructure to ensure it works well as a whole e.g., the connections, response time, etc.?
- How can we decrease human intervention in smart city management?
- How can we efficiently and automatically monitor and manage the resources within the smart city infrastructure, e.g., sensors, bandwidth, storage, CPU, memory, etc.?
- What is an appropriate architecture for an autonomic system that is an integral part of a smart city infrastructure?
- What are the autonomic services required for managing aspects of a smart city?

- Assuming a policy-based approach, what kinds of policies are needed to manage the smart city infrastructure?

7.1 Contributions of the Work

In the realm of smart city infrastructure, our research endeavors to address a series of pivotal questions and challenges. At the core of our work lies a commitment to advancing the understanding and management of smart city ecosystems, and in doing so, we have made several substantial contributions. Autonomic smart city management is a new topic and not enough work has been done in this field. A few researchers have focused on monitoring the performance of the smart city architecture itself and considered autonomic management for some aspects of the IoT network only, such as the network or cloud-based applications, and most of that has focused on monitoring the smart city environment such as temperature, pollution, etc. However, our purpose is to monitor not only the devices and sensors data and information in a smart city, but also the smart city infrastructure as well and we developed an autonomous system for the management of the network, services, processes, and applications. Using our approach, we could monitor the CPU and memory usage, response time, network status and so on.

Contributions of our research are:

First and foremost, we introduced a Comprehensive Model of a Smart City Infrastructure. While prior research has often focused on isolated aspects of smart city components, our model takes a holistic approach, capturing the entire spectrum of infrastructure elements - from sensors to hosts and applications. This holistic perspective allows us to delve into the complexities of managing the end-to-end infrastructure, providing valuable insights into how these various components interact and how they can be optimally managed.

Second, our approach does not operate in isolation; instead, it leverages the presence of existing components. With the ubiquity of IoT platforms and data analysis tools for sensor data, we have incorporated these components into our model. We assume the presence of an IoT platform for sensor management, a data filtering component for data quality enhancement,

and a performance monitoring component integrated into the autonomic management system. By embracing these tools, we ensure that our model aligns with current technological trends and practices within the smart city landscape.

Having a data filtering unit between the devices and the IoT platform that cleans the data before reaching the IoT platform is the first advantage of our approach. This unit is helpful because it prevents data redundancy in the databases and anomalies in the data. This process is important in smart city scenarios where we must manage a huge amount of data with limited resources. In addition to the aforementioned reason, we can apply machine learning techniques to the data in this unit.

In addition to infrastructure management, we have delved into the realm of real-time sensor monitoring and management. Our research places a strong emphasis on monitoring sensor attributes and measurements in real-time, providing invaluable insights that complement the offerings of IoT platforms. This real-time insight empowers decision-makers and ensures that smart city operations can be fine-tuned for maximum efficiency and effectiveness.

At the core of our research is the presentation of a Policy-Driven Autonomic Management Model, which is instrumental in streamlining and enhancing the efficiency and reliability of smart city operations. By introducing a policy-based approach, we empower smart cities to autonomously manage various aspects of their infrastructure, guided by established policies. Our approach to performance management leverages action policies to guide autonomic management decisions, defining a set of possible actions when specific objectives are violated. These objectives encompass not only ensuring quality of service requirements but also optimizing resource usage. Additionally, the use of configuration policies provides the ability to dynamically reconfigure components and applications, enabling systems to be easily deployable in diverse environments and under changing operational characteristics.

To bridge the gap between technical complexities and practical administration, we introduce a Management Interface designed for administrators. This user-friendly interface provides crucial insights to empower administrators with the information needed to ensure that smart city operations align seamlessly with high-level goals and objectives.

Demonstrating the real-world utility of our models, we have created prototypes for smart city infrastructure and an autonomic management system. These prototypes serve as examples, illustrating the practical advantages of our models and the real-world potential of adopting autonomic management in smart cities.

7.2 Limitations of the Work

In this section, we address some of the constraints and limitations of our work, shedding light on areas where improvements and future exploration may be necessary. These limitations are provided in the following.

Prototype: Our prototype was useful in demonstrating aspects of the overall approach, but it was limited in scope, limiting more extensive experimentation and evaluation. For example, in our prototype, we employed ThingsBoard as our IoT platform. However, there exists a plethora of alternative platforms suitable for integration within a city's IoT ecosystem. The ASCMS should possess the capability to accommodate this diversity in order to establish itself as a robust and adaptable solution.

Experiments: We considered a single smart city configuration, i.e., the sensors, etc. Additional environments could be constructed with different properties and configurations to enable more extensive experimentation and evaluation and to assess the scalability and performance of the prototype. While our work addresses infrastructure monitoring and performance management, it is important to note that the network configuration in a genuine smart city is considerably more intricate. In such environments, a multitude of network metrics necessitates monitoring and management due to the inherent complexity.

Scenarios and Policies: It is also important to note that the scenarios presented here may not encompass all potential occurrences within a smart city. The actual actions taken in a real smart city may vary from our suggestions. Nevertheless, this research serves as an initial stepping stone. Once appropriate policies are established, the system can adapt to address a broader spectrum of scenarios and potential failures that may arise in an authentic smart city environment.

Pattern Prediction: Our system lacks the capability to proactively anticipate metric patterns, particularly in the realm of performance metrics or some environmental factors. Furthermore, it lacks the proactive capability to allocate additional resources, specifically in the case of memory, before reaching a point of saturation. We can define a threshold to avoid saturation, but it is important to note that this threshold remains fixed until manually adjusted by the administrator.

Lack of Distributed Architecture: In our approach a centralized system is responsible for monitoring and managing the entire smart city ecosystem. While centralized systems offer operational efficiencies and streamlined management, they bring some limitations deserving of careful consideration. The specter of a single point of failure looms prominently, as any malfunction or downtime in the central node can disrupt the entire system. Additionally, the system's vulnerability to network dependencies exposes it to disruptions in connectivity. Security concerns heighten with the concentration of data in a centralized location, becoming an enticing target for unauthorized access. Recognizing these challenges, it becomes imperative to explore alternative options, such as distributed management, allowing for the collaboration and seamless communication of various management systems. This exploration could mitigate the identified limitations and contribute to more resilient and adaptable smart city infrastructure.

Moreover, our autonomic management system, much like many automation approaches, utilizes the ECA (Event-Condition-Action) format for rule storage. It is important to note that this format, while widely used, may pose limitations in expressing rules that do not align seamlessly with its structure.

Automated Rule Definition: The absence of automated rule definition and the system's inability to self-learn are also among the limitations of our work.

7.3 Future Directions

This research was only a starting point for autonomic management of the smart city infrastructure and there are several ideas that can be worked on as future directions. As

highlighted in the limitations section, there exist specific challenges that merit future attention and resolution.

The prototype used to evaluate the approach, while complex, is still limited in many ways. A more robust and more extensive prototype should be developed, perhaps based around interacting virtual machines. More extensive smart city "environments" would also be needed for experimentation, e.g., more sensors, networks, hosts, and applications to evaluate the scalability of the system. A further extension might be to build a "smart city configuration" in a lab with actual sensors, applications, etc. and have it managed by an extended prototype.

In the future we also need to measure the system's overhead to assess the efficiency of the management system, especially as it scales. This involves evaluating resource utilization, response times, and computational demands under varying conditions. Understanding the system's overhead will guide optimization efforts and ensure its adaptability and efficiency in accommodating the evolving demands of a growing smart city ecosystem.

The prototype should also be extended to operate in a distributed computing environment. While components already communicate, enhancing the prototype to function in a distributed manner would be a natural extension. This would then lead to considering how the model should be best mapped to a distributed architecture to enhance scalability and efficiency.

Applying machine learning techniques to the management system would be beneficial to proactively predict the variable patterns and future resources and prevent resource saturation before happening. Another extension for future work can be adding autonomic performance management to the cloud infrastructure.

Bibliography

1. A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, 14 February 2014.

2. J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. abs/1207.0203, 2013.

3. "IDC," [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prAP46737220. [Accessed: June 24, 2021].

4. U. Rosati and S. Conti, "What is a Smart City Project? An Urban Model or A Corporate Business Plan?," Procedia - Social and Behavioral Sciences, vol. 223, pp. 968-973, 10 June 2016.

5. T. Bakıcı, E. Almirall and J. Wareham, "A Smart City Initiative: The Case of Barcelona," Journal of the Knowledge Economy volume, vol. 4, p. 135–148, June 2013.

6. ISO, "Smart cities Preliminary Report 2014," ISO copyright office, Geneva, 2015.

7. S. Zygiaris, "Smart City Reference Model: Assisting Planners to Conceptualize the Building of Smart City Innovation Ecosystems," Journal of the Knowledge Economy, vol. 4, no. 2, p. 217–231, 08 March 2012.

8. J. M. Schleicher, M. Vögler, C. Inzinger and S. Dustdar, "Towards the Internet of Cities: A Research Roadmap for Next-Generation Smart Cities," in ACM First International Workshop on Understanding the City with Urban Informatics, 2015.

9. V. Fernandez-Anez, "Stakeholders Approach to Smart Cities: A Survey on Smart City Definitions," in Smart Cities, 2016.

10. C. Badii, E. Belay, P. Bellini, d. Cenni, M. Marazzini, M. Mesiti, P. Nesi, G. Pantaleo, M. Paolucci, S. Stefano, M. Soderi and I. Zaza, "Snap4City: A scalable IOT/IOE platform for developing Smart City Applications," 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 06 December 2018.

11. Snap4City, "Snap4City: Smart aNalytic APp builder for sentient Cities and IOT," [Online]. Available: https://www.snap4city.org/.

12. FIWARE Foundation, "FIWARE - Open APIs for Open Minds," [Online]. Available: https://www.fiware.org/.

13. "Orchestra Cities," [Online]. Available: https://www.orchestracities.com/.

14. T. Zahariadis, Z. Papadakis, F. Alvarez, A. Gonzalez, F. Lopez, F. Facca and Y. Al-Hazmi, "FIWARE Lab: Managing Resources and Services in a Cloud Federation Supporting Future Internet Applications," in 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, London, 2015.

15. "FIWOO | the FIWARE-based IoT Platform | IoT Solutions," [Online]. Available: https://www.fiwoo.eu/en/.

16. "The most simple enterprise IoT platform," [Online]. Available: https://thethings.io/.

17. "OpenIoT," [Online]. Available: http://www.openiot.eu/.

18. J. Kim and J.-W. Lee, "OpenIoT: An open service framework for the Internet of Things," 2014 IEEE World Forum on Internet of Things (WF-IoT), 24 April 2014.

19. D. Puiu, P. Barnaghi, R. Tonjes, D. Kumper, M. I. Ali, A. Mileo, J. Xavier Parreira, M. Fischer, S. Kolozali, N. Farajidavar, F. Gao, T. Iggena, T.-L. Pham, C.-S. Nechifor,

D. Puschmann and J. Fernandes, "CityPulse: Large Scale Data Analytics Framework for smart cities," IEEE Access, vol. 4, pp. 1086 - 1108, 2016.

20. M. A. Da Cruz, G. A. Marcondes, J. J. Rodrigues, P. Lorenz and P. R. Pinheiro, "Performance Evaluation of IoT Middleware through Multicriteria Decision-Making," in 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, 2019.

21. A. Luis Bustamante, M. Patricio and J. Molina, "Thinger.io: An Open Source Platform for Deploying Data Fusion Applications in IoT Environments," Sensors, vol. 19, no. 5, p. 1044, 01 March 2019.

22. "Thinger.io – Open Source IoT Platform," [Online]. Available: https://thinger.io/.

23. "Industrial IoT Messaging and Device Management Platform," [Online]. Available: https://github.com/mainflux/mainflux.

24. "Mainflux IoT Platform," [Online]. Available: https://www.mainflux.com/cloud.html.

25. J. Jo, J. Cho, R. Jung and H. Cha, "IoTivity-Lite: Comprehensive IoT Solution In A Constrained Memory Device," in 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2018.

26. "IoTivity," [Online]. Available: https://iotivity.org/.

27. [Online]. Available: https://github.com/iotivity/iotivity-lite.

28. P. Kostelnik, M. Sarnovsky and K. Fur, "The Semantic Middleware for Networked Embedded Systems Applied in the Internet of Things and Services Domain," Scalable Comput. Pract. Exp., vol. 12, 2011.

29. M. Eisenhauer, P. Rosengren and P. Antolin, "A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems," in 2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, Rome, 2009.

30. "LINKSMART® - FREE, OPEN SOURCE IOT PLATFORM," [Online]. Available: https://linksmart.eu/.

31. J. Gascon-Samson, M. Rafiuzzaman and K. Pattabiraman, "ThingsJS: towards a flexible and self-adaptable middleware for dynamic and heterogeneous IoT environments," in M4IoT '17: Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things, 2017.

32. "ThingsJS," [Online]. Available: https://github.com/DependableSystemsLab/ThingsJS.

33. "ThingsBoard - Open-source IoT Platform," ThingsBoard, 2021. [Online]. Available: https://thingsboard.io/.

34. J. Santos, T. Wauters, B. Volckaert and F. D. Turck, "Resource Provisioning for IoT application services in smart cities," in 2017 13th International Conference on Network and Service Management (CNSM), Tokyo, 2018.

35. C. McClelland, "IoT Device Management: What Is It and Why Do You Need It?," Leverege, 26 February 2019. [Online]. Available: https://www.leverege.com/blogpost/what-is-iot-device-management-why-you-need-it.

36. L. Gürgen and S. Honiden, "Management of Networked Sensing Devices," Taipei, 2009.

37. M. Ersue, D. Romascanu, J. Schoenwaelder and U. Herberg, "Management of networks with constrained devices: Problem statement and requirements," Internet Engineering Task Force (IETF), vol. RFC 7547, May 2015.

38. M. Aboubakar, M. Kellil and P. Roux, "A review of IoT network management: Current status and perspectives," Journal of King Saud University - Computer and Information Sciences, 02 April 2021.

39. "OpenZipkin. A distributed tracing system," [Online]. Available: https://zipkin.io/.

40. "Prometheus - Monitoring system & time-series database," [Online]. Available: https://prometheus.io/.

41. "OpenCensus," [Online]. Available: https://opencensus.io/.

42. "Jaeger: open-source, end-to-end distributed tracing," [Online]. Available: https://www.jaegertracing.io/.

43. "OpenTelemetry," [Online]. Available: https://opentelemetry.io/.

44. "The OpenTracing Project," [Online]. Available: https://opentracing.io/.

45. "What is Datadog?Definition from SearchITOperations," [Online]. Available: https://searchitoperations.techtarget.com/definition/Datadog.

46. "Cloud Monitoring as a Service | Datadog," [Online]. Available: https://www.datadoghq.com/.

47. "New Relic | Monitor, Debug and Improve Your Entire Stack | New Relic," [Online]. Available: https://newrelic.com/.

48. "Grafana: The open observability platform | Grafana Labs," Grafana, [Online]. Available: https://grafana.com/. [Accessed: May 28, 2023].

49. "Dynatrace | The Leader in Automatic and Intelligent Observability," [Online]. Available: https://www.dynatrace.com/.

50. "Gartner | Delivering Actionable, Objective Insight to Executives and Their Teams," [Online]. Available: https://www.gartner.com/.

51. "Application Observability | Applications and Microservices," Dynatrace, [Online]. Available: https://www.dynatrace.com/platform/applications-microservices-monitoring/. [Accessed: Nov 05, 2022].

52. P. Sakhardande, S. Hanagal and S. Kulkarni, "Design of disaster management system using IoT based interconnected network with smart city monitoring," in 2016 International Conference on Internet of Things and Applications (IOTA), 2016.

53. S. Suakanto, S. H. Supangkat, Suhardi and R. Saragih, "Smart city dashboard for integrating various data of sensor networks," in International Conference on ICT for Smart Society, 2013.

54. C.-C. Usurelu and F. Pop, "My City Dashboard:Real-time Data Processing Platform for Smart Cities," Journal of Telecommunication and Information Technology, pp. 89-100, 2017.

55. M. Dryjanski, M. Buczkowski, Y. Ould-Cheikh-Mouhamedou and A. Kliks, "Adoption of Smart Cities with a Practical Smart Building Implementation," IEEE Internet of Things Magazine, vol. 3, no. 1, pp. 58-63, 2020.

56. L. T. De Paolis, V. De Luca and R. Paiano, "Sensor data collection and analytics with thingsboard and spark streaming," in 2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS), 2018.

57. Y. Chen and D. Han, "Water quality monitoring in smart city: A pilot project," Automation in Construction, vol. 89, pp. 307-316, 2018.

58. A. F. Rachmani and F. Y. Zulkifli, "Design of IoT Monitoring System Based on LoRa Technology for Starfruit Plantation," in TENCON 2018 - 2018 IEEE Region 10 Conference, Jeju, Korea, 2018.

59. M. Aarthi and A. Bhuvaneshwaran, "IoT based drainage and waste management monitoring and alert system for smart city," Annals of the Romanian Society for Cell Biology, vol. 25, no. 3, pp. 6641-6651, 2021.

60. S. Jha, L. Nkenyereye, G. P. Joshi and E. Yang, "Mitigating and Monitoring Smart City Using Internet of Things," Computers, Materials & Continua, vol. 65, no. 2, pp. 1059-1079, 2020.

61. N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar and S. Mahmoud, "SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services," IEEE Access, vol. 5, pp. 17576-17588, 24 July 2017.

62. W. Apolinarski, U. Iqbal and J. X. Parreira, "The GAMBAS middleware and SDK for smart city applications," in 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), Budapest, 2014.

63. M. Handte, P. José Marroń, G. Schiele and M. Serrano Matoses, Adaptive Middleware for the Internet of Things – The GAMBAS Approach, 2019.

64. CityPulse," [Online]. Available: https://github.com/CityPulse/CityPulse-City-Dashboard.

65. E. Manoel, M. J. Nielsen, A. Salahshour, S. S. K. V. L. and S. Sudarshanan, Problem Determination Using Self-Managing Autonomic Technology, San Jose: IBM International Technical Support Organization, 2005.

66. J. Kephart and D. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, pp. 41 - 50, 14 January 2003.

67. R. Sterritt, M. Parashar, H. Tianfield and R. Unland, "A concise introduction to autonomic computing," Advanced Engineering Informatics, vol. 19, no. 3, pp. 181-187, 2005.

68. M. Parashar and S. Hariri, "Autonomic Computing: An Overview," Lecture Notes in Computer Science, vol. 3566, pp. 257-269, 2005.

69. R. Calinescu, "General-Purpose Autonomic Computing," Autonomic Computing and Networking, pp. 3-30, 2009.

70. L. Gurgen, O. Gunalp, Y. Benazzouz and M. Galissot, "Self-aware cyber-physical systems and applications in smart buildings and cities," 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), 06 May 2013.

71. A. E. Braten, F. A. Kraemer and D. Palma, "Autonomous IoT Device Management Systems: Structured Review and Generalized Cognitive Model," IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4275 - 4290, 15 March 2021.

72. D. Kyriazis and. T. Varvarigou, "Smart, autonomous and reliable Internet of Things," Procedia Computer Science, vol. 21, pp. 442-448, 2013.

73. L. Gurgen, A. Cherbal, R. Sharrok and S. Honiden, "Autonomic Management of Heterogeneous Sensing Devices with ECA Rules," in 2011 IEEE International Conference on Communications Workshops (ICC), Kyoto, 2011.

74. R. Sharrock, A. Cherbal, L. Gürgen, T. Monteil and S. Honiden, "Thinking Autonomic for Sensing Devices," in 2010 Sixth International Conference on Autonomic and Autonomous Systems, Cancun, 2010.

75. L. Broto, D. Hagimont, P. Stolf, N. Depalma and S. Temate, "Autonomic management policy specification in Tune," in Proceedings of the 2008 ACM symposium on Applied computing - SAC '08, 2008.

76. S. T. Arzo, R. Bassoli, F. Granelli and F. H. Fitzek, "Multi-Agent Based Autonomic Network Management Architecture," IEEE Transactions on Network and Service Management, vol. 18, no. 3, pp. 3595 - 3618, September 2021.

77. E. Mezghani, S. Berlemont and M. Douet, "Autonomic Coordination of IoT Device Management Platforms," in 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, 2020.

78. N. Ayeb, E. Rutten, S. Bolle, T. Coupaye and M. Douet, "Coordinated autonomic loops for target identification, load and error-aware Device Management for the IoT," in

2020 15th Conference on Computer Science and Information Systems (FedCSIS), Sofia, 2020.

79. J. Kosińska and K. Zieliński, "Autonomic Management Framework for Cloud-Native Applications," Journal of Grid Computing, vol. 18, no. 4, p. 779–796, 26 September 2020.

80. G Saadon, Y Haddad, M Dreyfuss, N Simoni. “Decision-making support for an autonomous software-defined network orchestrator.” IET Networks, vol 11,no1, p. 13-26 Jan 2022

81. A. N. Lam, O. Haugen and J. Delsing, "Dynamical Orchestration and Configuration Services in Industrial IoT Systems: An Autonomic Approach," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 128-145, 2022, doi: 10.1109/OJIES.2022.3149093.

82. H. V. Sampaio, F. Koch, C. B. Westphall, R. D. Boing, and R. N. Cruz, "Autonomic Management of Power Consumption with IoT and Fog Computing," arXiv preprint arXiv:2105.03009, 2021.

83. K. Nalinaksh, P. Lewandowski, M. Ganzha, M. Paprzycki, W. Pawłowski, and K. Wasielewska-Michniewska, "Implementing Autonomic Internet of Things Ecosystems – Practical Considerations," in V. Malyshkin (Ed.), Parallel Computing Technologies. PaCT 2021, Lecture Notes in Computer Science, vol. 12942, Springer, Cham, 2021, pp. 442-454. https://doi.org/10.1007/978-3-030-86359-3_32.

84. N. F. S. d. Sousa and C. E. Rothenberg, "CLARA: Closed Loop-based Zero-touch Network Management Framework," in 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Heraklion, Greece, 2021, pp. 110-115. doi: 10.1109/NFV-SDN53031.2021.9665048.

85. J. Kosińska and K. Zieliński, "Experimental Evaluation of Rule-Based Autonomic Computing Management Framework for Cloud-Native Applications," in IEEE

Transactions on Services Computing, vol. 16, no. 2, pp. 1172-1183, 1 March-April 2023, doi: 10.1109/TSC.2022.3159001.

86. L. E. Villela Zavala, A. Ordoñez García and M. Siller, "Architecture and Algorithm for IoT Autonomic Network Management," 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Atlanta, GA, USA, 2019, pp. 861-867, doi: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00155.

87. A. Riker, R. Mota, D. Rośario, V. Pereira, and M. Curado, "Autonomic management of group communication for Internet of Things applications," Int. J. Commun. Syst., vol. 35, no. 11, p. e5200, 2022. doi: 10.1002/dac.5200.

88. B. K. Singh, M. Danish, T. Choudhury, and D. P. Sharma, "Autonomic Resource Management in a Cloud-Based Infrastructure Environment," in T. Choudhury, B. K. Dewangan, R. Tomar, B. K. Singh, T. T. Toe, and N. G. Nhu (Eds.), "Autonomic Computing in Cloud Resource Management in Industry 4.0," EAI/Springer Innovations in Communication and Computing, Springer, Cham, 2021, pp. 225-236. https://doi.org/10.1007/978-3-030-71756-8_18.

89. P. Shukla, P. Richhariya, B. K. Dewangan, T. Choudhury, and J. S. Um, "The Architecture of Autonomic Cloud Resource Management," in T. Choudhury, B. K. Dewangan, R. Tomar, B. K. Singh, T. T. Toe, and N. G. Nhu (Eds.), "Autonomic Computing in Cloud Resource Management in Industry 4.0," EAI/Springer Innovations in Communication and Computing, Springer, Cham, 2021, pp. 183-193. https://doi.org/10.1007/978-3-030-71756-8_14.

90. M. Mangla, S. Deokar, R. Akhare, and M. Gheisari, "A Proposed Framework for Autonomic Resource Management in Cloud Computing Environment," in T. Choudhury, B. K. Dewangan, R. Tomar, B. K. Singh, T. T. Toe, and N. G. Nhu (Eds.), "Autonomic Computing in Cloud Resource Management in Industry 4.0,"

EAI/Springer Innovations in Communication and Computing, Springer, Cham, 2021, pp. 117-125. https://doi.org/10.1007/978-3-030-71756-8_10.

91. R. S. M. L. Patibandla, V. L. Narayana, and A. P. Gopi, "Autonomic Computing on Cloud Computing Using Architecture Adoption Models: An Empirical Review," in T. Choudhury, B. K. Dewangan, R. Tomar, B. K. Singh, T. T. Toe, and N. G. Nhu (Eds.), "Autonomic Computing in Cloud Resource Management in Industry 4.0," EAI/Springer Innovations in Communication and Computing, Springer, Cham, 2021, pp. 127-137. https://doi.org/10.1007/978-3-030-71756-8_11.

92. Q. Zhou, A. J. Gray, and S. McLaughlin, "SeaNet -- Towards A Knowledge Graph Based Autonomic Management of Software Defined Networks," arXiv preprint arXiv:2106.13367, 2021.

93. C. Lin, H. Khazaei, A. Walenstein, and A. Malton, "Autonomic Security Management for IoT Smart Spaces," ACM Trans. Internet Things, vol. 2, no. 4, Article 27, 20 pages, November 2021. https://doi.org/10.1145/3466696.

94. E. Okhovat and M. Bauer, "Monitoring the Smart City Sensor Data Using Thingsboard and Node-Red," 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI), Atlanta, GA, USA, 2021, pp. 425-432, doi: 10.1109/SWC50871.2021.00064.

95. "Node-RED", Nodered.org, 2021. [Online]. Available: https://nodered.org/.

96. "ThingsBoard architecture", ThingsBoard, 2021. [Online]. Available: https://thingsboard.io/docs/reference/.

97. "Beach Water Quality- Automated Sensors", DataHub, 2021. [Online]. Available: https://datahub.io/JohnSnowLabs/beach-water-quality---automated-sensors.

98. "Host Definition | IoT ONE Digital Transformation Advisors," IoT ONE, [Online]. Available:

https://www.iotone.com/term/host/t273#:~:text=Formal,a%20company's%20data%20processing%20process. [Accessed: April 10, 2023].

99. "Use Resource Monitor to monitor storage performance | TechRepublic," TechRepublic, [Online]. Available: https://www.techrepublic.com/article/use-resource-monitor-to-monitor-storage-performance/. [Accessed: Sep 29, 2022].

Curriculum Vitae

Name: **Elham Okhovat**

Post-secondary Education and Degrees:
The University of Western Ontario
London, Ontario, Canada
2018-2023 Ph.D.

Shiraz University
Shiraz, Fars, Iran
2012-2015 M.Sc.

Honours and Awards:
WGRS scholarship
2018-2022

First ranked among B.Eng. students
2011

Related Work Experience

Teaching Assistant
The University of Western Ontario
2018-2022

Research Assistant
The University of Western Ontario
2018-2023

Web Developer and Administrator
Zamen Salamati Institute
2018

Founder, Webmaster and Website Developer
Aparatech E-Commerce Assistant Co.
2016- 2018

Webmaster and Website Designer and Developer
Dr.Web Security Lab
2016

Website Administrator and Network Marketing Manager
Ghasr Talaee International Hotel (Golden Palace Hotel)
2015-2016

Order Manager and Network Engineer
Agah Brokerage Co.
2014-2015

Website Designer and Development Manager
Bartarandishan IT Academy
2013-2014

Publications:

E. Okhovat and M. Bauer, "Monitoring the Smart City Sensor Data Using Thingsboard and Node-Red," in *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, Atlanta, 2021.