sensors

Article

Situation-Aware IoT Data Generation towards Performance Evaluation of IoT Middleware Platforms

Shalmoly Mondal [1], Prem Prakash Jayaraman [1,*], Pari Delir Haghighi [2], Alireza Hassani [1] and Dimitrios Georgakopoulos [1]

1 School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Hawthorn 3122, Australia
2 Department of Human-Centred Computing, Monash University, Clayton 3800, Australia
* Correspondence: pjayaraman@swin.edu.au

Abstract: With the increasing growth of IoT applications in various sectors (e.g., manufacturing, healthcare, etc.), we are witnessing a rising demand of IoT middleware platform that host such IoT applications. Hence, there arises a need for new methods to assess the performance of IoT middleware platforms hosting IoT applications. While there are well established methods for performance analysis and testing of databases, and some for the Big data domain, such methods are still lacking support for IoT due to the complexity, heterogeneity of IoT application and their data. To overcome these limitations, in this paper, we present a novel situation-aware IoT data generation framework, namely, SA-IoTDG. Given a majority of IoT applications are event or situation driven, we leverage a situation-based approach in SA-IoTDG for generating situation-specific data relevant to the requirements of the IoT applications. SA-IoTDG includes a situation description system, a SysML model to capture IoT application requirements and a novel Markov chain-based approach that supports transition of IoT data generation based on the corresponding situations. The proposed framework will be beneficial for both researchers and IoT application developers to generate IoT data for their application and enable them to perform initial testing before the actual deployment. We demonstrate the proposed framework using a real-world example from IoT traffic monitoring. We conduct experimental evaluations to validate the ability of SA-IoTDG to generate IoT data similar to real-world data as well as enable conducting performance evaluations of IoT applications deployed on different IoT middleware platforms using the generated data. Experimental results present some promising outcomes that validate the efficacy of SA-IoTDG. Learning and lessons learnt from the results of experiments conclude the paper.

Keywords: IoT; IoT middleware platforms; benchmarking; Fuzzy Situation Inference; IoT data generation; situation transition; performance evaluation

check for updates

Citation: Mondal, S.; Jayaraman, P.P.; Delir Haghighi, P.; Hassani, A.; Georgakopoulos, D. Situation-Aware IoT Data Generation towards Performance Evaluation of IoT Middleware Platforms. *Sensors* **2023**, *23*, 7. https://doi.org/10.3390/s23010007

Academic Editor: Luigi Ferrigno

Received: 8 November 2022
Revised: 5 December 2022
Accepted: 14 December 2022
Published: 20 December 2022

1. Introduction

The rise in IoT applications is unparalleled in the last few years [1]. IoT middleware cloud platforms are used extensively for IoT deployments as they provide many functionalities such as managing IoT devices, real-time data processing, distributed data storage, among others. This complicates the already strenuous comparison of IoT middleware platforms, making it essential to have a standard solution that enables IoT application developers to compare different IoT middleware options best suited for their application. A key step in developing such a solution is to be easily conduct performance evaluation of IoT middleware platforms running the IoT applications. One key step in being able to achieve this is to be able to generate IoT data that is close to real-world data to undertake performance evaluation of IoT application running on different IoT middleware platforms and identify performance bottlenecks. Such data for IoT scenarios for evaluating the performance of IoT middleware platforms can either be real-world data from various real-world domains of IoT, which require access to large volumes of data generated by the IoT devices, or they can be simulated data that has been artificially generated which mimics real-world data, with known underlying patterns. While using real data in performance evaluation in not always recommended [2], it is difficult to replay the data sequentially with the real frequency. Additionally, real data may not always be available to represent real-world events such as earthquakes or road accidents. However, developers need access to data to test and debug their application and often rely on synthetic data suited for their needs which often does not properly represent real-world scenarios, which motivates the need for efficient data generation techniques. While domains such as databases have well-established standard datasets [3,4], and Big Data domain has some emerging data sets [5–7], IoT still lacks a methodology to generate comprehensive data that can capture the properties of IoT applications (e.g., real-time dynamic data)and hence can be used to evaluate the performance of IoT middleware platforms. To generate data while keeping the significant features of real IoT data, it is important to understand the different needs and requirements for IoT applications. However, relatively little effort has been given to considering the application requirements to generate data. Additionally, data generators in the existing literature do not properly represent the data generated by IoT devices, which are usually in the form of continuous stream of data. One such example is the frequency with which the data should be generated. RIoT Bench [8] for example, addresses the conventional data analytic tasks that are carried out over real-time IoT data streams. However, the data are not representative of event-based data, which is generated in most IoT applications, since IoT applications are mostly event or situation driven.

IoT applications are often associated with real-world situations. The notion of 'Situations' is used to represent any real-world events, activities or situations such as 'road accidents', 'fire', 'walking'. It has been established from the literature that sensory data combined with contextual information can help in better understanding of the environment [9,10]. With the evolution of IoT, and the increasing use of such IoT applications, it is crucial to have an additional layer of contextual information to generate IoT data corresponding to real-time situations. For example, let us consider the example of an IoT application for real-time monitoring of traffic conditions of a road segment. The situations associated with such an application that describe real-world scenarios can be: 'low_traffic', 'moderate_traffic', and 'high_traffic'. Sensory data such as location, speed, density, and trip time, combined with context information such as location, and weather can be used to define the situations. The defined situations can then be used as a baseline to generate real-world IoT data. Moreover, in real-world, the situations gradually transition from one situation to another. For example, a situation of 'low_traffic' gradually transition to 'moderate_traffic' and then to 'high_traffic'. Capturing such details to generate data makes data more realistic.

Existing research efforts in IoT data generation have used different methods. Some of them are discussed below: TPCx-IoT [11] benchmark generated data based on the Yahoo Cloud Serving Benchmark framework (YCSB) [12]. The dataset is based on data from sensors from an electric power substation. The authors in [13,14] used machine learning based models to generate time series smart meter data. However, there is a prevalent gap in the literature since data has not been generated considering real-world situations, and their transitions. To address the aforementioned challenges, in this paper, we propose a novel framework, SA-IoTDG, to generate data that is similar to real-world data and takes into consideration the IoT application requirements, the real-world situations and their transitions. Hence, rather than randomly generating data without considering the IoT application requirements, we aimed to address the need for simulating situation aware data for IoT applications. Two key components of proposed SA-IoTDG are: (1) a data generation tool for generating sensor data, keeping the significant features of real data, and (2). A SysML model that captures the requirements of IoT applications. This model is an extension to our past work [15]. We extend this model to capture the IoT application situations and the transitions. Our proposed framework allow users to define their application entities, the situations for their application, and generate IoT data. This enables them to perform simulations prior to having the actual dataset. In addition, most IoT applications, rely on supervised learning algorithms, where, training the dataset is essential. The accuracy of these algorithms depends highly on the training and test process. It is very difficult to find a comprehensive training dataset that gives high accuracy. Hence, in this paper, we use a Fuzzy Logic based situation theory approach to generate relevant IoT data that does not require training the dataset.

Our approach to defining situations for IoT applications and generating data to test and evaluate IoT middleware platforms running IoT applications is based on Fuzzy Situation Inference (FSI) [16], which combines Fuzzy Logic and Context Spaces theory. While Fuzzy Logic provides the flexibility for reasoning and considers the uncertainties and inaccuracies of a situation for an IoT application, Context Spaces theory provide an additional layer of contextual information to provide enhanced reasoning about real-world situations. In [16], the authors state the usage of FSI for situation inference under uncertainty. The situation transitions are handled by a Markov chain-based model integrated into the framework. The proposed framework can help both researchers and IoT application developers to generate IoT data for their application. The generated dataset can be pushed to different middleware platforms to compute relevant benchmarks. The tool is extensible, and can be configured and tailored to user specific needs and requirements. The innovation of the system lies in its ability to deal with the uncertainty and transition between different situations and generate data in near-real-time that reflects the dynamic properties of IoT data.

The contributions of the paper are as follows:

- We propose a novel situation based data generation framework which serves as a starting point for identifying performance bottlenecks of IoT deployments. The framework is configurable and can be extended for multiple IoT use cases.
- We extend the IoTSySML model [15] to represent IoT situations and transitions.
- We introduce a Markov chain-based approach for situation transition to generate real-world IoT data to support performance evaluation of IoT middleware platforms running IoT applications.
- We evaluate the data generation capability of the framework by simulating a traffic monitoring application that validates the dynamic data generation as situation transitions occur.

The outline of this paper is as follows: Section 2 gives an overview of the related work. Section 3 introduces the basic concepts required for this work. Section 4 presents the proposed framework for situation aware data generation. Section 5 demonstrates the implementation of the proposed framework. Section 6 presents the experimental evaluations and the analysis. Finally, Section 7 concludes the paper.

2. Related Work

As discussed above, IoT middlewares are being used extensively in IoT. However, there is limited literature in IoT Middleware benchmarks and how the different middlewares can be compared with each other. The most notable work in this area is of [17,18] where the authors have compared two publish-subscribe based middlewares, FIWARE and M2M, based on a set of qualitative and quantitative metrics. The middleware platforms have been considered as a black box without considering the internal implementation. Other works [19,20] that compare middleware platforms are limited to conventional metrics such as reliability, scalability, price, and availability. However, the complexity of IoT applications gives rise to the need for metrics tailored for IoT domain. For example, Medvedev et al. [21], evaluated the performance of OpenIoT platform from data ingestion and data storage perspective. The authors in [22] use contextual queries for benchmarking IoT context management platforms. In another research by Salhofer et al. [23], the different components (General Enablers) of the FIWARE platform has been thoroughly analyzed and evaluated from an application deployment point of view. The authors reported that IoT application could benefit by utilizing the various services offered by the platform. However, the lack of updated documentation makes the process of application development difficult.

Cruz et al. [24] presents a performance evaluation study of a few existing middleware platforms and establish qualitative and quantitative metrics to evaluate the performance of the middleware platforms, such as InatelPlat, Konker, Linksmart, Orion+STH and Sitewhere. There have also been a few works on evaluating the performance of Fiware platform to investigate its usability in various domains of IoT. For example, Araujo et al. [25] evaluated the performance of FIWARE with respect to scalability to report its usage in smart city applications. Similarly, another study [26] was carried out to explore the applicability of FIWARE platform in the smart farming domain. Table 1 summarizes the related work in this area. From the table, we can see that benchmark studies in IoT are fragmented. Existing research in this area targets the different layers of the IoT ecosystem. For instance, ref. [27] is a device level benchmark for IoT edge devices. Similarly, while TPCx-IoT [28] targets the IoT gateways, RIoTBench [8] evaluates the stream processing capabilities for IoT applications. However, it is limited to DSPS applications that are centrally hosted in the cloud. In addition, ref. [29] targets cloud-based applications. Hence, while standard holistic benchmarks exist in other related areas such as big data [5,7,30–32], databases [3,4], RDF stream processing [33,34], the same cannot be said for IoT.

Table 1. Summary of existing IoT benchmarks.

Paper	Benchmarking Targets	Evaluation Metrics	IoT Application Requirements	IoT Data Generation
[27]	IoT device		✗	✗
[35]	IoT Gateways	IoTps [1]	✗	smart meter data
[36]	IoT Middleware Platform	IoTps [1]	✗	smart meter data
[8]	IoT middleware platform	latency, throughput, CPU, memory utilization	✗	✗
[21]	IoT middleware platform	IPS [2], resource utilization	✗	✗
[23]	IoT middleware platform		✗	✗
[22]	IoT middleware platform	Contextual Queries	✗	✗
Our work	IoT middleware platform	IPS, DID [3], Query Response Time	✓	✓

[1] Performance Metrics, [2] Ingests Per Second, [3] Data Ingestion Delay.

It can be established from the literature [5,31–33,36,37] that a dataset is an important factor for conducting performance evaluation of a system. However, there is limited work around real world data generation in IoT targeted for performance evaluation of IoT middleware platforms. Some of the prominent works in the area are IoTSim [38] which aims to analyse IoT applications, but is limited to Mapreduce models. The other notable work is of Cardoso et al. [17] where the dataset used is data from a traffic monitoring scenario that publishes the average speed of traffic in each street of the city on an hourly basis. Another research [39] proposed a dataset that targets Intrusion Detected Systems for IoT and IIoT application domain. Iftikhar et al. [14] used a predictive based model to generate time series smart meter data. Similarly, Liu et al. [13] generate smart meter datasets using a regression model. There are a few simulators that have been used in the IoT space, most of which are applicable to cloud and Wireless Sensory Network (WSN). Full stack simulators such as Cloudsim [40], the OASIS standard Devices Profile for Web Services (DPWSSim) [2], and iFogSim [41] have been used to simulate IoT application scenarios. iFogSim is limited to fog and edge computing. Simulators such as IoTSim [38] and SimIoT focus on the data processing of IoT applications. While IoTSim is restricted to simulating MapReduce programming model, SimIoT is currently unable to simulate different types of sensors. Simulators such as OMNET++ [42], Cooja [43], NS2 [44] are network simulators and have been widely used in WSN. However, these simulators hardly provide support for performance evaluation in IoT.

As stated earlier, our key objective is to use the generated data for evaluating the performance of different IoT middleware platforms. So, we want data to be generated that mimics real-data. To achieve this, we might take into consideration the requirements of IoT applications and other details, for example, real-world situations that they are associated with. For example, in a fire monitoring application, the severity of a fire can be classified into situations such as 'low threat', 'medium threat' or 'high threat' and actions can be taken accordingly. We have reviewed the literature to investigate the methods used for modelling and representing situations of IoT applications. In [45], the authors propose a domain specific language for representing IoT applications. Some of the works based on context modelling technique have defined situations using a form of predicate calculus [46]. Pre-defined rules are triggered when a situation occurs and situations are inferred based on the result set 'true', 'false' or 'possibly true'. Other context modelling approaches focused on the temporal aspects of a situation [10,47]. This work was further extended to address the uncertainty aspect of IoT [48]. Other works leveraged Fuzzy Logic-based approaches to represent the uncertainty and dynamic situations of IoT applications [49,50]. However, just using Fuzzy Logic to represent IoT applications was not enough, and not suitable for IoT environments. Hence, it can be seen in the literature that Fuzzy Logic has been integrated into Context Spaces theory as a formal method of representing situations for ubiquitous environment, which is more suitable for IoT applications [16,51]. IoT applications based on real-world situations can benefit using FSI modelling, as it enables generating data according to the application requirements. Additionally, FSI captures the minor and gradual changes which occurs in real-world situations. As an example, in the fire monitoring application discussed above, if the fire threat is 'low', the fire situation can be handled by just sending a warning message, whereas if the fire threat is 'high', actions to put out the fire and an alarm should be issued. In this way, the false alarm rate can be decreased. Without an FSI approach, there are no intermediate values, but absolute values: Is there a fire? yes/no or 0/1. Based on the FSI modelling technique, a set of rules are used to define Situations. These FSI rules are then triggered to infer a situation. Multiple rules can be used to define a situation [16]. However, most of these existing approaches do not consider the transition between these situations. In real-world situations, gradually transition from one state to another. If we take the same traffic monitoring scenario described above, a situation of low_traffic gradually transitions to moderate_traffic. Capturing these granular details enables generating a more realistic, looking dataset. Based on the literature, Markov chain is one of the most common approach that has been used to handle state transitions [52,53]. Some approaches [54–56] combine Markov chain and Fuzzy Logic to take into consideration the uncertainties while computing the transition probability matrix by replacing the crisp transition probabilities by fuzzy numbers.

In summary, benchmarks in IoT are under researched and there is limited research work around generation for evaluating the performance of IoT middleware platforms. Table 1 illustrates that none of the existing studies take into consideration the IoT application requirements in generating data that is similar to real world data. IoTAbench [36] is the closest to our work, which addresses our goal partially. The authors have developed a Markov chain-based synthetic data generator for their framework. While the data generation approach is interesting and generates realistic time-series data, the data generator does not consider IoT application requirements to generate relevant data. We have highlighted in Table 1 how our proposed framework will address the existing gaps in the literature including (a) devising a template for specifying IoT application requirements, (b) generate IoT data taking into consideration application's requirements. Furthermore, to the best of our knowledge, there is no data generator in the IoT domain which considers IoT situations and their transitions for generating IoT data. The IoT data simulator is a more recent attempt [57], that generates data on the fly while also having the capability of replaying existing data with modified data. However, it does not provide support for simulating situation based IoT data, generating data for multiple situations or simulating the situation transitions.

3. Preliminaries

In this section, we discuss some of the concepts used for this research for representing situations and generating relevant IoT data.

3.1. Fuzzy Situation Inference Theory

The Fuzzy Situation Inference (FSI) Theory [16] can be used for representing and inferring real-world situations. FSI builds on the theory of Context Spaces [10] which is a formal approach for context modelling and reasoning context-aware applications. To overcome the limitations of this theoretical modelling approach, in dealing with uncertainty of real-world situations, the FSI approach combines Context Spaces with Fuzzy Logic-based approach to benefit from both approaches in one methodology. We will first give a brief overview of the Context Spaces Theory (CST) and then discuss the FSI theory.

The Context Space Theory (CST) [10] was developed as a formal approach for reasoning about context in multidimensional space, known as Situations. Some other basic concepts of CST are described below:

- **situation space**, where real-world situations are perceived as sub-spaces within a n dimensional application space.
- **context attributes**, which are the data generated from sensors that can be used for reasoning or inferring situations. Context attributes can be either measured by sensors directly, or derived from sensory data. For example, air temperature, light level, noise level, air humidity, etc., can be the context attributes for a smart office scenario.
- **region** which is a domain of allowed values for a context attribute.
- **context state** which represents a data point within an n application space at a point of time, where n is the number of context attribute.

The term situation is commonly used to represent a contextual state in an application space. There exists several real-world situations in the IoT application space, such as 'meeting' in a smart office scenario [48] or a situation of 'hypertension' in a health monitoring scenario [16]. Situation awareness is the specification of such complex real-world situations by taking into consideration its relationship in space and time [58].

In FSI, we use the term 'linguistic variable' to represent any contextual parameters. A situation is represented using FSI rules. The rules comprise linguistic variables, each of which constitute a fuzzy set. An FSI rule consists of an antecedent and a consequent, similar to fuzzy rules. The antecedent can be associated with more than one fuzzy set. FSI rules consists of multiple conditions joined by fuzzy operators. The most commonly used are an 'AND' or 'OR' operators. A single FSI rule has the form:

R01: If CA1 is A and CA2 is B, then the Situation is C

where the *'if'* part is the antecedent, and the *'then'* denotes the consequent. *CA1*, *CA2* and *CA3* are linguistic variables defined by fuzzy sets *A*, *B*, and *C*. These rules are evaluated using membership functions. Some of the commonly used membership functions are triangular function, trapezoidal function among others. FSI uses the notion of weights, and confidence to infer a situation [10]. Each variable in the rule is assigned a weight (a value between 0 and 1) which determines the importance of the variable in determining a situation. The consequent of the rule determines the degree of certainty with which a situation is occurring and is given by the below equation:

$$Confidence = \sum_{i=1}^{n} w_i \mu(x_i) \tag{1}$$

where $\mu(x_i)$ is the membership degree of ith element in the fuzzy set x, and $w_i\mu(x_i)$ is the weighted membership degree of the element x_i. A situation is said to occur if its degree of certainty tends to 1. We will discuss this in details in Section 4.3.

3.2. Markov Chain

In conditional probability theory, a Markov chain $X_n, n \geq 0$ is a stochastic model, whose state space S is a finite set, for $T = (0, 1, 2, \ldots)$. X_n denotes the state of the system at time n. It can be used to calculate the probability of moving from one state to another and is expressed as $X_n = i$, which denotes X_n being in state i. Markov chains are characterized by the Markov property, which states that the transition to a state at time t+1 depends only on the current state at time t. Figure 1 illustrates a generic Markov chain.

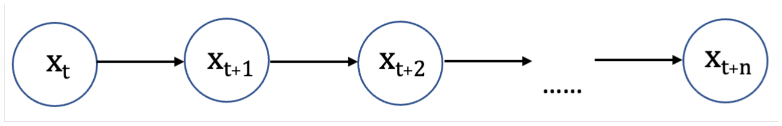


Figure 1. A Generic Markov Chain.

For any $j, x_0, x_1, \ldots, x_n$, the Markov property is defined by:

$$P(X_{n+1} = j | X_0 = x_0, \ldots, X_n = i_n) = P(X_{n+1} = j | X_n = i\} \tag{2}$$

The above equation (Equation. 2) emphasizes that the probability of X_{n+1} depends only on the probability of X_n. The idea is that given the present state, the past states have no influence on the future state or simply put, the future state depends only on the present state. Hence, the Markov chain is 'memory less', since the probability of transitioning to a different situation depends on the current situation, but not on the past situations. The multitude of these probabilities to transition from one state to another is represented in a matrix form, called a transition probability matrix P, where $P = [P_{ij}]$, where P_{ij} is the probability of moving from state i to state j. The rows represent the current state and the columns represent the next state. The probability of X_{n+1} being in state j, given that X_n is in state i, is denoted by $P_i^n j^{n+1}$, i.e.,

$$P_i^n j^{n+1} = Pr\{X_{n+1} = j | X_n = i\} \tag{3}$$

The above equation explains that the transition probabilities are not only a function of initial and final states, but also of the time of transition.

4. Situation Aware IoT Data Generation Framework

This section presents the SA-IoTDG (Situation Aware IoT Data Generation) framework that can be used to define situations, simulate the transition between the situations and generate IoT data based on the order of the occurrence of the situations. In the following subsections, we start with a high-level system overview, followed by the key components of the proposed framework.

4.1. System Overview

As shown in Figure 2, the system broadly comprises SA-IoTDG, our proposed framework (gray dotted box), communication channels, and IoT middleware platforms to run IoT applications. Users provide their application requirements ① to the framework to generate IoT data. The generated data can be sent to different cloud managed IoT middleware platforms over various communication channels ⑦.

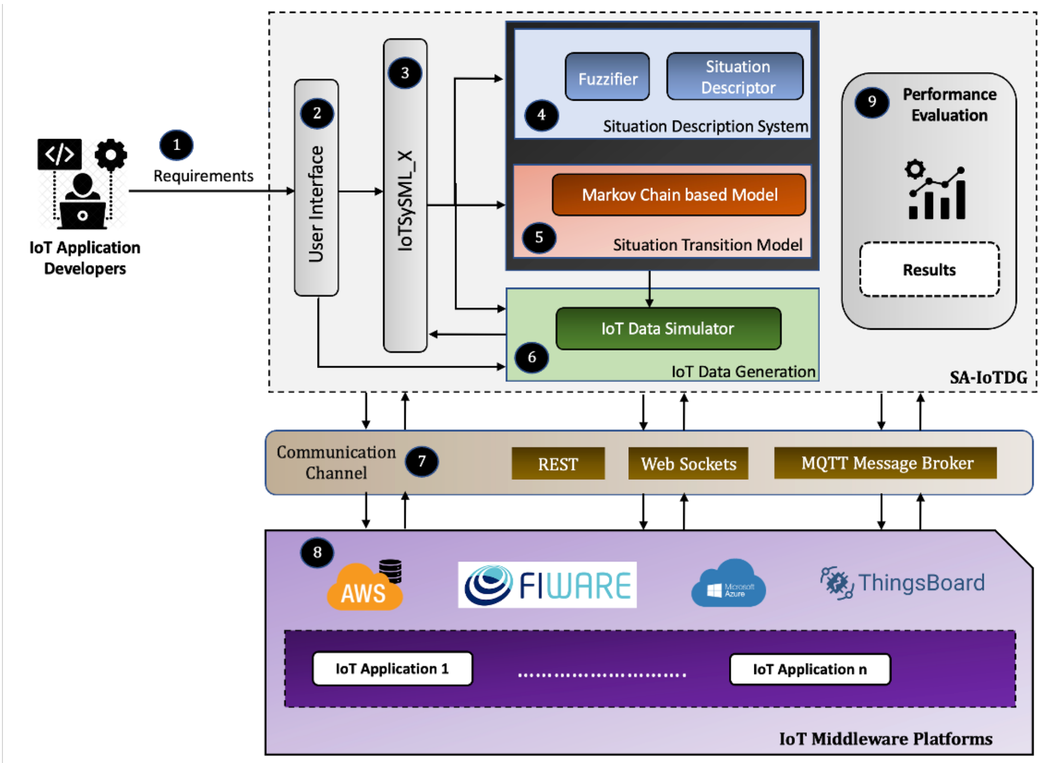


Figure 2. High-level System Overview.

SAIoT-DG Architecture: SAIoT-DG comprises a user interface ② to register the situations and run simulations, IoTSySML_X ⑤, a domain reference model to capture user specific application requirements, components to model the situations ④ and their transitions ⑤, and a component to generate the data ⑥. We describe briefly the key components for developing the SA-IoTDG framework. The framework is divided into four major parts:

- *IoTSySML_X:* We extend on our previous IoTSySML model [15] which was developed to capture the requirements of IoT applications. We integrate the concepts of situations and context from the domain of CST [10] to strengthen the model.
- *Situation Description System:* We define real-world situations of IoT application using an FSI [16]-based approach.
- *Situation Transition Model:* We handle the various transitions between the situations. The transitions have been modelled using Markov Chain concepts [53,59].
- *IoT Data Generation:* We focused on generating data with respect to the defined situations. The generated data should mimic the real-time IoT data as much as possible. Another essential feature we wanted was to enable users give their desired configuration. By configuration, we mean a set of parameters that controls the data being generated. An example is a parameter to declare the frequency of the data generation.

  In the following subsections, we discuss each of these components in details.

4.2. IoTSySML_X

In this section, we extend on our previous model [15] to enhance its functionality. Our previous model IoTSySML [15] was developed to provide a formal way to capture the requirements of IoT applications, which was missing in existing literature. Identifying and modelling the application requirements enables generating data relevant to the IoT applications. For this paper, we have extended this model to include situation theory, Context Spaces and Fuzzy Logic concepts. By adding these new elements, we are providing support for more context to represent an IoT application scenario and their requirements in a better way. For example, we represent IoT applications, using situations from real-world scenario. If we take the same example of the IoT traffic monitoring scenario discussed above, the application can be represented by situations of 'high_traffic', 'low_traffic', or 'moderate_traffic'. Adding contextual information such as location (e.g., parking areas), weather conditions (e.g., snow or rain), or road conditions (e.g., road work, accidents) provides granular details which enable realistic data generation for evaluating the performance of IoT middleware platforms. The model has been conceived using a UML class, where we extend existing UML concepts using Stereotypes. A concept is encapsulated using a 'block'. Each block is a stereotyped (customized) extension of a UML Class and has a set of properties and defined relation with other blocks. We first explain the extension we applied to the traditional UML. Figure 3 shows the stereotypes that we used for the model. The stereotype ***'Context'*** has been extended from existing UML model. The 'Context' class represents the context information of any IoT application scenario and is defined by elements *'ContextSpace'*, *'ContextualParameter'*, and *'ContextElements'*. The ContextSpace stereotype represents the situations or events a context element might be involved in. ContextElement describes the entity itself in a situation. A context element can be a user, or any physical object. ContextualParameter represents the observable properties from an IoT device. Similarly, the stereotype ***'Situation'*** has been extended from existing UML and entails all information related to defining a situation and is defined by stereotypes *'SituationProperty'* and *'FSIElements'*. SituationProperty represents the characteristics of a situation. For example, the transitions a situation might go through. FSIElements represent the conditions applied to defining a situation.

We now illustrate the class diagram of the domain concepts depicted in Figure 4. The green elements represent concepts from our previous model [15]. The purple boxes indicate the extended concepts. The *'IoT devices'* element is the virtual representation of a physical IoT device. IoT device class comprises *'Sensor'*, which makes *'Observation'*, a class provided by SOSA [60] to capture a sensor observation. It represents the data generated by the sensors and have attributes: *'Phenomenon Time'* to capture the temporal aspect of an observation as a window or in intervals, and *'Values'* to represent the observations made by the sensors (for example, acceleration values generated by accelerometer or temperature values generated by temperature sensors). Values are represented as a key-value pair. A geolocation observation for example has two key value pairs of latitude, longitude and can have a type which can have value such as circle. Every sensor is associated with *'metadata'* [61], which has attributes type and value. An example of metadata to represent the unit of a temperature sensor data is : 'type' = "string', 'value' = "Celsius'.

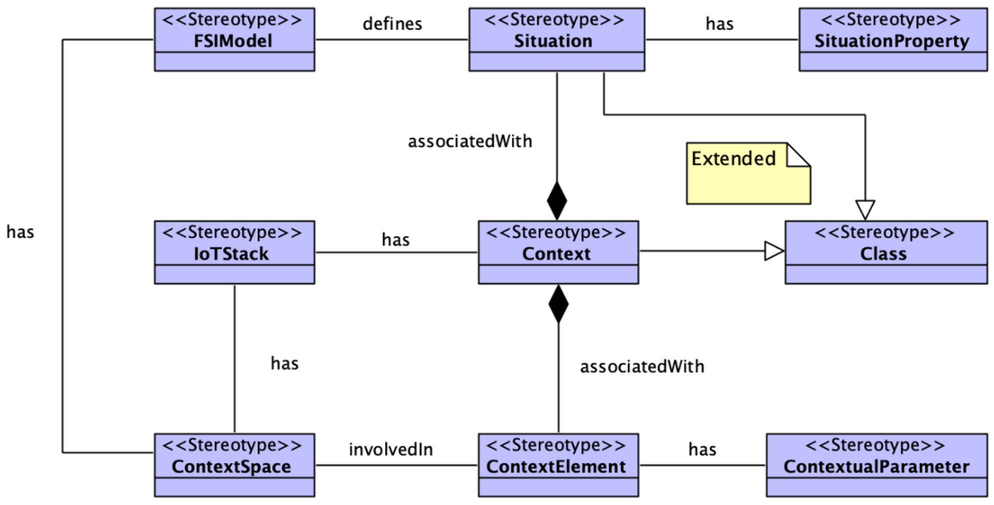


Figure 3. Extended class stereotypes.

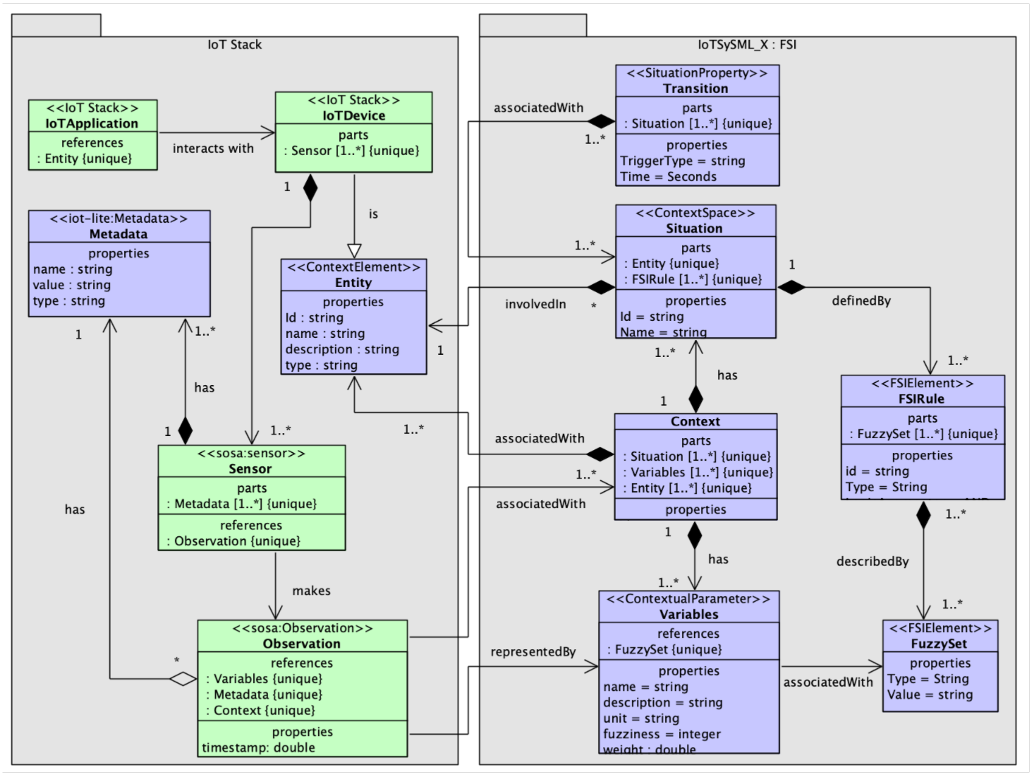

IoT Stack
IoTSySML_X : FSI
<<IoT Stack>> IoTApplication
references : Entity {unique}
interacts with
<<IoT Stack>> IoTDevice
parts : Sensor [1..*] {unique}
<<iot-lite:Metadata>> Metadata
properties name : string value : string type : string
is
<<ContextElement>> Entity
properties Id : string name : string description : string type : string
has
<<sosa:sensor>> Sensor
parts : Metadata [1..*] {unique}
references : Observation {unique}
makes
<<sosa:Observation>> Observation
references : Variables {unique} : Metadata {unique} : Context {unique}
properties timestamp: double
<<SituationProperty>> Transition
parts : Situation [1..*] {unique}
properties TriggerType = string Time = Seconds
associatedWith
<<ContextSpace>> Situation
parts : Entity {unique} : FSIRule [1..*] {unique}
properties Id = string Name = string
involvedIn
definedBy
<<FSIElement>> FSIRule
parts : FuzzySet [1..*] {unique}
properties id = string Type = String
Context
parts : Situation [1..*] {unique} : Variables [1..*] {unique} : Entity [1..*] {unique}
properties
describedBy
<<ContextualParameter>> Variables
references : FuzzySet {unique}
properties name = string description = string unit = string fuzziness = integer weight : double
representedBy
<<FSIElement>> FuzzySet
properties Type = String Value = string

Figure 4. UML class diagram of the domain concepts.

'Entities' are a representation of any physical object or individual, and are involved in *'Situations'*. 'Situations' are any real-world situation such as 'low traffic' or any activity such as 'working'. They are characterized by ID, name, timestamp, definition, previous state and certainty (probability with which a situation is occurring). An example of an entity 'worker' being involved in a situation would be (id=S_1, name = 'cutting meat', prev_state = 'idle', certainty = 0.7). A situation comprises a set of attributes involved in defining the situation at a particular time. These attributes have been modelled as a *'Context'* class and refers to characteristics or information associated with 'Sensor Observation'. It also entails data and behavior of the entity class [48]. Hence, we see a relationship between Entity and Context. Additionally, we can observe that every instance of 'Context' has an association relation with *'Variables'* which refers to characteristics or information associated with 'Observation'. Table 2 summarizes the relationships between the different classes in the model. Variable are stereotyped as 'ContextualParameters' and are characterized by name, definition, weight (the relative importance of the parameter in inferring a situation) and is associated with a *'FuzzyRule'* element stereotyped as *'FSIElelements'*. Another FSIElement is the element Fuzzy Rule, which is involved in defining a situation. A situation can be defined using multiple 'Variables'. Similarly, there can be multiple FSIRule instances to define one instance of any situation. The *'Transition'* element allows the situation class to move from one state to another and is stereotyped as a feature of the situation class. To allow the transitions, three types of triggers were considered: time-constraint, rule based, and external environmental factors. Each of these has been captured by the *'Triggers'* class.

4.3. Situation Description System

In this section, we describe our situation description system. The components of the system are a fuzzifier, a fuzzy inference engine, a rule base, and a situation reasoner. Figure 5 illustrates these components. We explain the process of FSI based situation definition, with our traffic monitoring motivating scenario.

Table 2. Description of Relationships.

Relationship	Description	Base Class	Target Class
Composition	Objects that are associated with each other can not remain in the scope of a system without each other. The target class cannot exist without the base class i.e..if target class is deleted, base class gets deleted	Metadata Sensor Entity Situation Situation Variables FuzzySet	Sensor IoT Device Situation Context Transition Context FSIRule
Aggregation	Objects that are associated with each other can remain in the scope of a system without each other	Metadata Metadata	Sensor Observation
Generalization	Represents an "is-a" relationship	Entity	IoTDevice
Association	Semantic relationship between classes representing a logical connection between them	IoTApplication Observation Variables	IoTDevice Variables FuzzySet

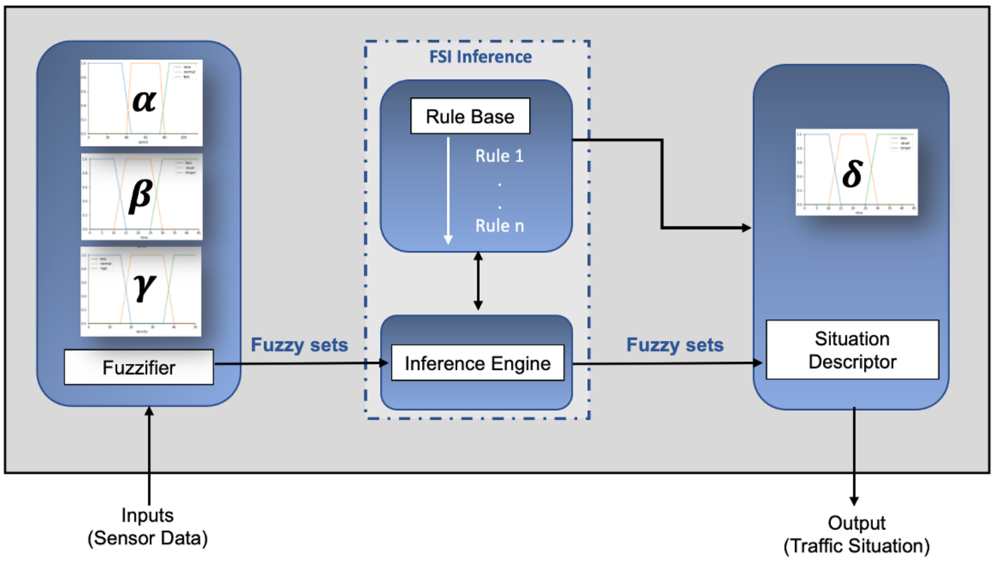


Figure 5. FSI based Situation Description System.

4.3.1. Fuzzifier

Fuzzification entails defining the input and output parameters to the FSI system to determine the traffic condition of a road segment. The fuzzifier receives the crisp context information as input from the sensors. In FSI, these crisp inputs are converted to a fuzzy set by the fuzzifier. Fuzzy sets are stipulated by their membership functions, $\mu_A(x)$ which determined how an input is mapped to membership values between 0 and 1 given by Equation (4)

$$x_k = \mu_A(x_k) \tag{4}$$

where x_k is the input data and $\mu_A(x_k)$ is the is the degree of membership of an element x in the fuzzy set A for every $x \epsilon A$.

A traffic monitoring scenario discussed in Section 1 can be involved in situations such as 'low_traffic', 'moderate_traffic', and 'high_traffic'. We use three linguistic variables to define these traffic situations. These variables and their fuzzy subsets are described below:

Speed, α: Denotes the average speed of the cars on the road segment.

$$\alpha = \begin{cases} 1 & \text{slow, } 0 \geq \alpha \geq 50 \\ 2 & \text{normal, } 40 \geq \alpha \geq 80 \\ 0 & \text{fast, } 70 \geq \alpha \geq 120 \end{cases} \tag{5}$$

Density, β: Average density or the number of cars on a road segment.

$$\beta = \begin{cases} 1 & \text{low, } 0 \geq \beta \geq 15 \\ 2 & \text{normal, } 10 \geq \beta \geq 35 \\ 0 & \text{high, } 30 \geq \beta \geq 45 \end{cases} \tag{6}$$

Trip time, γ: Average time taken to cross the road segment.

$$\gamma = \begin{cases} 1 & \text{less, } 0 \geq \gamma \geq 15 \\ 2 & \text{usual, } 10 \geq \gamma \geq 30 \\ 0 & \text{longer, } 25 \geq \gamma \geq 40 \end{cases} \tag{7}$$

4.3.2. FSI Inference

The FSI Inference component consists of a rule base and an Inference Engine. It receives the contextual parameters as inputs from the fuzzifier component and uses them to evaluate the FSI rules. A rule base in an integral part of an FSI based inference system. These fuzzy rules are defined by domain experts and are stored in a Rule repository. The rules are subjective and rely on the discernment of domain experts. An example of an FSI rule is shown below:

R01: If A AND B AND C, THEN X

where A, B and C are rule conditions to define a Situation X. As an example, the condition A can be written as 'speed is slow', B is another condition 'density is 'high'. Similarly, the variable C can be written as 'trip_time is longer'. These conditions denote Situation X is occurring. To simplify, the same rule can be written as:

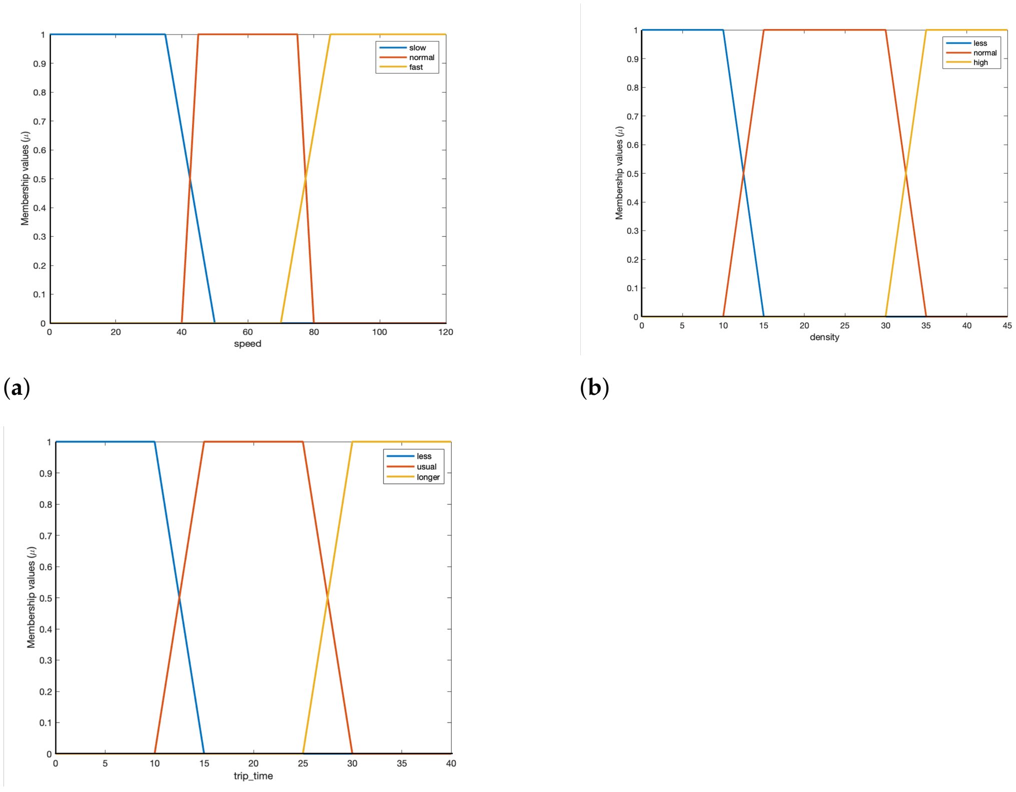
R01: If speed is 'slow' AND density is 'high' AND trip_time is 'longer', THEN situation is 'High Traffic'

Table 3 shows a subset of the defined FSI rules for our traffic monitoring scenario.

Table 3. FSI rule base for a traffic monitoring scenario.

	Input		**Output**
IF	AND	AND	THEN
A	B	C	X
speed is fast	density is less	trip_time is less	Low Traffic
speed is normal	density is less	trip_time is less	Low Traffic
speed is normal	density is normal	trip_time is usual	Moderate Traffic
speed is slow	density is high	trip_time is longer	High Traffic

The inference engine evaluates these pre-defined rules to define a situation. We formulated the rules based on an extensive analysis of IoT traffic monitoring scenario applications [62–65]. We adopt the Mamdani fuzzy inference algorithm [66] for defining our situations for IoT applications. We have categorized each input variables into different fuzzy subsets. We represent the fuzzy subsets by a trapezoidal membership function. Figure 6 shows that an input variable can belong to two fuzzy subsets simultaneously, with varying degrees of membership. A single $x \epsilon X$ may be associated with multiple fuzzy subsets. As an example, the for the input linguistic variable speed, a crisp value of 45 km/h can either belong to fuzzy subset 'slow' or 'normal' (Figure 6). The membership function assigns a degree of similarity to these values. If applied to the 'slow' fuzzy subset, the membership function might assign a degree of 0.2 ($\mu_{slow}(45) = 0.2$). The same value, when applied to the 'normal' fuzzy set, might have a degree of 0.9 assigned ($\mu_{normal}(45) = 0.9$). This signifies, a speed of 45 km/hr has greater chances of belonging to the fuzzy subset normal.



(**a**) (**b**) (**c**)

Figure 6. Fuzzy subsets for the input variables of a traffic monitoring application. (**a**) α. (**b**) β. (**c**) γ.

4.3.3. Situation Descriptor

Each situation is associated with a confidence value, which is calculated using:

$$Confidence = \sum_{i=1}^{n} w_i \mu(x_i) \tag{8}$$

We illustrate the final step for defining the situation 'Low Traffic' for the below rule:

R01: If speed is 'normal' AND density is 'less' AND trip_time is 'less', THEN situation is 'Low Traffic'

Table 4 shows the input values of the linguistic variables speed, density and trip_time. Each of these variables have been assigned a weight, which indicates its relevance in defining a situation. For example, we have assigned a weight of 0.4 to 'speed' and 'density' which denotes both these variables have the same importance in defining the situation 'low_traffic'. Additionally, these variables have been mapped to fuzzy subsets and assigned a degree of membership by a trapezoidal membership function (Equation (4)) depicted by the $\mu_x(i)$ column.

Table 4. Example of calculating the confidence value for the occurrence of a situation 'Low Traffic'.

Input Variables	Values	Conditions	Weights	$\mu_x(i)$
speed	45	speed is normal	0.4	0.9
density	15	density is less	0.4	0.3
time	10	trip_time is less	0.2	0.5

We calculate the confidence value below using Equation (8):

$$\text{Confidence} = w_\alpha . \mu_{normal}(45) + w_\beta . \mu_{less}(15) + w_\gamma . \mu_{less}(10)$$
$$= (0.4 * 0.9) + (0.4 * 0.3) + (0.2 * 0.5) = 0.58$$

The situation descriptor defines the occurrence of a situation based on this confidence value. From the confidence value, we can say that there is 0.58 probability that the situation Low Traffic is occurring. Hence, the FSI based model discussed above gives a mapping between the inputs (α, β AND γ) and the output (δ) and hence define the situations.

4.4. Situation Transition Model

After situations have been defined using FSI, in this section, we present our proposed situation transition model which has been conceptualized using Markov chains [67].

Markov models can be represented by state machines, where the transition of each state can be modeled as a continuous-time Markov process as shown in Figure 7, where P_i and P_j are the probabilities of transitioning from state i to state j. These transitions between the states generates a sequence of states over time. For example, Figure 7 shows a Markov model with two states, $S = S_1, S_2$. The possible sequence of states could be:

$$\{S_1, S_2, S_1, S_1, S_1, S_2, S_2, S_1, S_2, S_2, \dots\}$$

The probability of transitioning to the next state can be estimated by performing experiments and can be expressed as a transition probability matrix. The transition probability matrix can be determined based on experimental data, such as from a measured sequence of events [56]. In Figure 7, if there is a transition of state S_i to state S_j, the transition probability matrix can be calculated by [56], where

$$\pi_{ij} = \frac{N_{ij}}{N_{oi}} \tag{9}$$

π_{ij} is the probability of transitioning from state S_i to state S_j, N_{ij} is the number of observed transitions from state S_i to state S_j, and $N_{io} = \sum_{j=1}^{M} N_{ij}$, is the total amount of data that belongs to state S_i i.e., the total number of transitions that are initiated from state S_i.

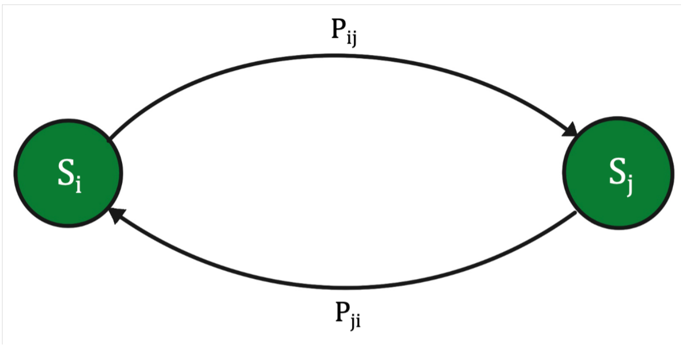


Figure 7. Two State Markov Chain.

We explain our proposed model with our traffic monitoring motivating scenario and define some of the concepts used for this research below:

1. **Situation Space** $S = \{S_1, S_2, \ldots, S_n\}$: corresponds to the possible situations of an IoT application. For our scenario the situation space would be:
   (a) *low_traffic*
   (b) *moderate_traffic*
   (c) *high_traffic*
2. **Transition Kernel** = $k(S_n|S_{n+1})$: denotes the probability of transition from situation S_n to situation S_{n+1} given that situation S_n has already occurred, with certain restrictions shown below:
   - → $k(low\_traffic|low\_traffic) = p$, probability of staying in the same situation, which is semantically equivalent to moving to the same state
   - → $k(low\_traffic|moderate\_traffic) = 1 - p$, probability of moving to moderate_traffic from low_traffic
   - → $k(low\_traffic|high\_traffic) = 0$, no direct transition from low to high_traffic
   - → $k(moderate\_traffic|low\_traffic) = p$, probability of making a transition to low_ traffic to moderate_traffic
   - → $k(moderate\_traffic|moderate\_traffic) = 1 - p^2$,, probability of staying in the same state of moderate_traffic
   - → $k(moderate\_traffic|high\_traffic) = 1 - p^2$, probability of moving to high_traffic from moderate_traffic
   - → $k(high\_traffic|low\_traffic) = 0$, no direct transition from high to low_traffic
   - → $k(high\_traffic|moderate\_traffic) = p$, probability of making a transition to high_traffic to moderate_traffic
   - → $k(high\_traffic|high\_traffic) = 1 - p$, probability of staying in the same high_traffic situation

   As seen in the above conditions, we restrict the direct transition of situation low_traffic to a situation of high_traffic or a direct transition from high_traffic situation to low_traffic situation. Simply put, we cannot transition from low_traffic to high_traffic situation in a single step, we need two steps in making the transition. The two-step transition probability is given by matrix P^2:

$$P(S_2 = j|S_0 = i) = P(S_{n+2} = j|S_n = i) = P^2_{ij} \tag{10}$$

   By Equation (2), the two-step probability for our traffic scenario is represented as:

$$P(low\_traffic|high\_traffic) = p(low\_traffic, moderate\_traffic)$$
$$p(moderate\_traffic, high\_traffic = P^2$$

3. **State Space** $s = \{InitialState, TransitionState, TargetState\}$: corresponds to the possible states in a situation transition sequence:
   (a) *initial state* —corresponds to the starting state for a transition sequence;
   (b) *transition state*—refers to the intermediate states a situation goes to before transitioning to the target state;
   (c) *target state*—corresponds to the final state a situation transitions into.
4. **Target Triggers**: corresponds to the triggers that might result in a situation transition. For this work, we have considered two triggers:
   (a) time-based—situations transition according to a time specified by users. After the threshold time is crossed, situation S_1 transitions to situation S_2 and continues the current execution, and
   (b) probability based—based on the probabilities given, situations transition from one state to another
5. **Situation Transition Probability Matrix** tPr: denotes the probability with which a situation would move to another. The elements of the transition matrix tPr are defined as:

$$(tPr)_{ij} = P_{ij} = P(S_1 = j|S_0 = i) = P(S_{n+1} = j|S_n = i)$$

for any n, where P_{ij} is the probability of making a transition from state 'i' to state 'j'.

Assuming there are n situations defined in our Situation Space which are denoted by $S_1, S_2, \ldots S_n$, and we represent each situation with a state in the Markov model, we will have n the number of states. The situations gradually transition from one state to another, i.e., there is a probability of the situations happening in a sequence $P(S_i, S_j \ldots Sn)$. The transition probability from the ith to jth state is given by a probability transition matrix P_{ij}. Hence, a Markov chain having n possible states, has a $n * n$ transition probability matrix given by:

$$P = \begin{bmatrix} P_{11} & P_{12} & \ldots & P_{1n} \\ P_{21} & P_{22} & \ldots & P_{2n} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ P_{n1} & P_{n2} & \ldots & P_{nn} \end{bmatrix}$$

Our system contains a finite set of states, and we represent the states by a finite state machine, where the transition of each state has been modeled as a continuous-time, finite-state Markov process. For our IoT traffic monitoring motivating scenario, we assume we have three situations, where each situation is modelled as a Markov state illustrated in Figure 8. However, our framework can be configured to add more situations if required. The user interface developed as part of the framework enables users to add situations as required by their IoT application. The diagram outlines the probabilities associated with making a transition from one situation to another. For example, it shows that there is a 50% chance to move from S_3 to S_2, and a 30% chance to transition to S_3 from S_1.

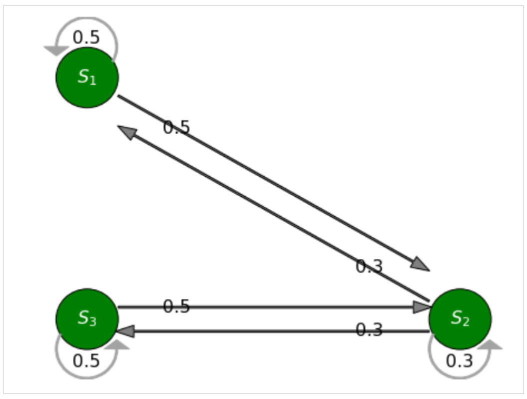


Figure 8. Three State Markov Chain.

Each state or situation in the Markov model has been characterized using Fuzzy Situation Inference Theory, as discussed in Section 4.3. There might be multiple situations occurring at the same time, with varying degrees of confidence. Hence, our proposed approach leverages Markov chains and incorporates fuzzy based reasoning to enhance the prediction accuracy of the situation transitions.

Next Situation Prediction

The execution of the situation transition model is given by a Markov model in the form of a deterministic finite automata (DFA) shown in Figure 8. The transitions in a Markov chain are probabilistic rather than deterministic, which means that there is no absolute certainty about what situation will occur at time $t + 1$. Each situation S_i is associated with a Markov state and the transition probability matrix determines the probability of transitioning between the states, i.e., between the situations $S_i, i = 1, \ldots, n$. The transition probability of the occurrence of a traffic situation at time $t + 1$, given the current traffic situation at time t, can be written as:

$$siTPr = P(S_t|S_{t+1}) \tag{11}$$

As stated above, we have three situations for our scenario, the transition model for these situations has been illustrated in Figure 8. A Markov chain model can be described completely based on two key factors:

Transition Probability: We express the transition probabilities in the form of a transition matrix below. For this experiment, for convenience we assume, equally likely probabilities to transition from one situation to another. Hence, the next situation prediction model is given by

$$S_{n+1} = S_n * tPr \tag{12}$$

where tPr = (P_{ij}) is the situation transition probability matrix given by:

$$\text{tPr} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}$$

The above transition probability matrix, tPr describes the traffic situation of a road segment and has the following properties:

(i) *Property 3.1* The matrix is a stochastic matrix, i.e., P_{ij} in a row should add to 1

(ii) *Property 3.2* If $P_{ij} \geq 0$, then the situation S_n can transition to S_{n+1}

The transition probability matrix defines the probability of transitioning from one situation to another. Hence, the next situation can be determined as a function of the current state and the transition probability, as given by Equation (12). Additionally, the probability of staying in the same state is semantically equivalent to moving to the same state, which are shown by self loops in Figure 8. Furthermore, if situation S_1 moves to situation S_3, and passes another transition state s_2, the probability of going to s_3 after n steps can be determined by:

$$p(S_1, S_3) = p(S_1, S_2) * p(S_2, S_3) = p^2 \tag{13}$$

An Initial State Vector: This is a probability vector which initializes a Markov chain and denotes the probability distribution of the initial states, i.e., each element in the vector represents the starting probability of a transition sequence at a given state. We initialize our Markov model and denote the initial state probability vector by:

$$P_{initial} = (p_1, p_2, p_3)$$

The initialization can be conducted in multiple ways, we might assume a fixed starting point, or we can allocate probabilities for starting the Markov process. For our work, we assume equally likely probabilities for starting the sequence of transitions, where our initial state probability vector would be $P_{initial} = (0.33, 0.33, 0.33)$ since we have three states in our Markov model.

4.5. IoT Data Generation

The IoT Data Generation is a key component of SA-IoTDG framework, which handles the data generation aspect. It has been built on the existing IoT data simulator stack [57]. Since, we aim to generate IoT data for the purpose of evaluating IoT middleware platforms, we would ideally want the generated data to be similar to real-world IoT data. Hence, to enable a realistic data generation, we consider the context of IoT applications, so the data generated is more close to real world IoT data. To achieve this, we have extended the functionality of the IoT data simulator to enable data generation for situation aware IoT applications. To elucidate further, as seen in the previous subsections, we define our situations leveraging FSI theory (Section 3.1), and specify the transition between these situations based on a Markov model (Section 4.4). The data generation component receives inputs from these modules: component ④ and ⑤ (Figure 2) and generates data based on information received from these components. Hence, the framework is able to adapt dynamically to the changing configuration, and generate data based on that.

Furthermore, IoT applications might have different requirements in terms of generating data. Some applications might require data to be generated more frequently than others. For example, a smart farming application [68] might require data from soil sensors every few minutes to monitor the soil quality. On the other hand, our traffic monitoring motivating scenario might require data to be generated every few seconds to obtain the real-time location and speed information of vehicles. Hence, rather than randomly generating data, we aimed to address the need for simulating situation aware data for IoT applications by considering the IoT application requirements. As an example, for this work we conceptualized an IoT traffic monitoring scenario, and generated a traffic dataset. The entity here is a road segment under consideration. The generated data mimics data from on road camera sensors in the particular road segment and consists of the average speed of the vehicles on a particular road segment, the vehicle count, and the estimated travel time. The frequency of the generation of the observations can be configured by the user. Furthermore, since most IoT smart city applications are location aware, we generate the data with geolocation coordinates of the users. For our experiment, we have generated observations with a sampling frequency of 1/5 (i.e., observations every 5 s). The data generation component is capable of generating data in near-real-time. We demonstrate the entity, the IoT devices involved in such a scenario, and the data they generate. SSN [69] and SOSA [70] ontologies are often used to represent sensor data observations. Figure 9 shows a sample ontological representation of generated traffic data observations using SOSA ontology. The data generated by the framework correspond to a one traffic data. We generated data with a sampling frequency of 1/5 s, to align it with real-world traffic data.

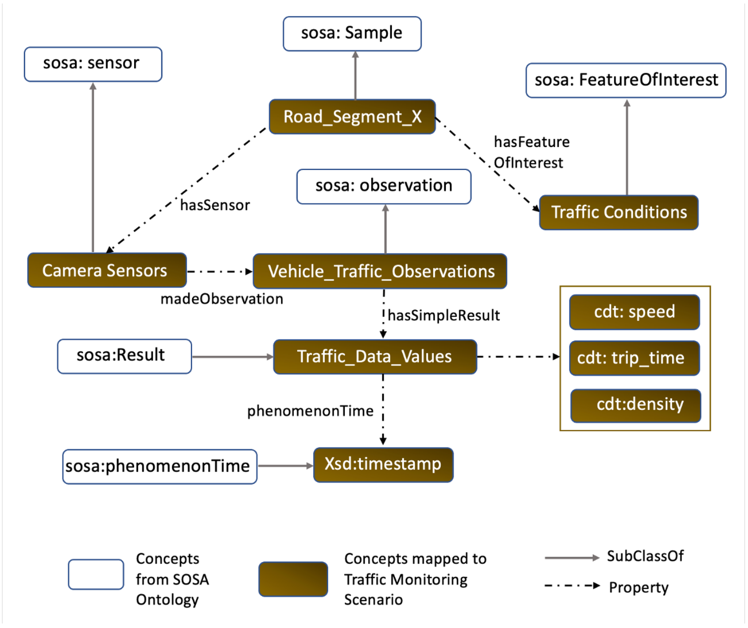


Figure 9. Illustration of the traffic sensor data.

5. Implementation of SA-IoTDG

In this section, we discuss the implementation details of the proposed framework (SAIoTDG) and explain how an application developer can use the framework to generate data relevant to his IoT application using sequence diagrams. SAIoTDG allows users to define situations corresponding to their IoT application with respect to the situation description system discussed in Section 4.3, which is illustrated in Figure 10. SAIoTDG also provides a functionality for users to configure the sequence of the situation transitions, which is illustrated in Figure 11. When the order and occurrence of a known situation is detected, the framework starts generating IoT data for the situation according to the configuration set by the user (sense frequency, number of data records to be generated).

At the beginning, 'users' (IoT Application Developers) enter their application requirements to IoTSySML_X which is an extension of our previous work [15]. The SysML model returns an XML specification document of the application requirements, which is first parsed into intermediate JSON and then a template is generated containing all the required details to register a situation. In the next step, the *Situation Manager* validates and stores the situation details in the *Situation Repository*. (Step 2 and 2.1) We used a MongoDB database to store the situation specific details. When a request is made to obtain the situation details (Step 3), a connection is established with the Situation Repository, that connects to MongoDB service, and fetches the situation details as a response in JSON format (Step 3.1).

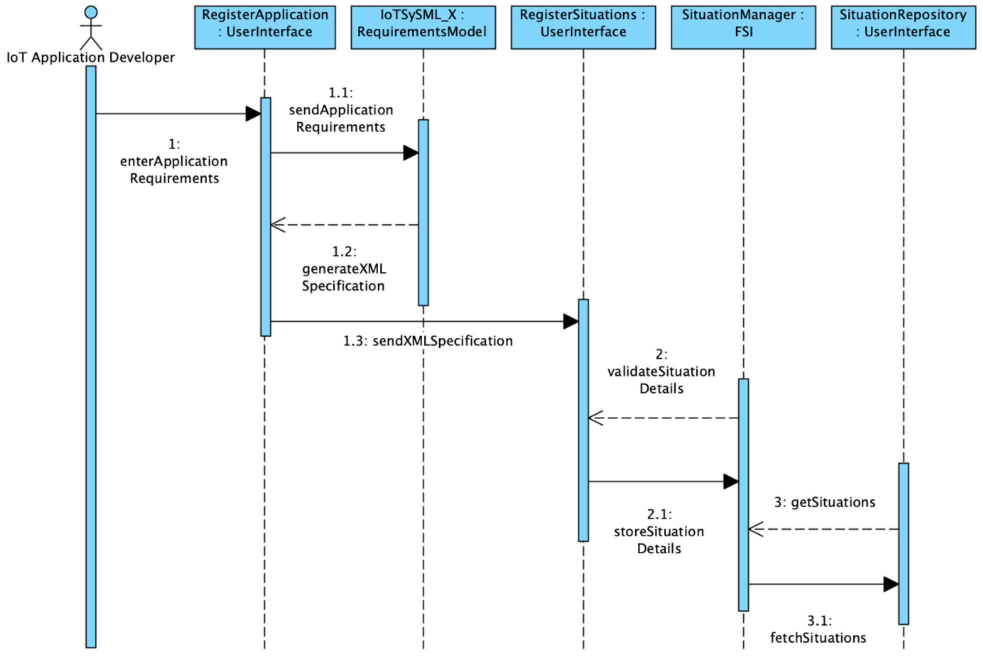

IoT Application Developer
RegisterApplication : UserInterface
IoTSySML_X : RequirementsModel
RegisterSituations : UserInterface
SituationManager : FSI
SituationRepository : UserInterface
1: enterApplication Requirements
1.1: sendApplication Requirements
1.2: generateXML Specification
1.3: sendXMLSpecification
2: validateSituation Details
2.1: storeSituation Details
3: getSituations
3.1: fetchSituations

Figure 10. Registering situations.

After successfully, registering the situations, users can specify the situation transition sequence to generate data. The workflow for running the simulation is illustrated in Figure 11. First, users need to specify the order of the situation transitions (Step 1). At this step, the parameters for data generation can be configured (Step 2). Once the desired sequence of transitions and the configurations have been selected, we establish a web-socket communication for generating the data. Parameters can be updated (Step 2.2) and data are generated with the new settings in near-real-time as the situation transitions occur. We use web sockets in FAST API (python web framework) for instantaneous data transfer without HTTP overhead. The generated data are then displayed to the user (Step 2.2.1) and then data can be sent to IoT middleware platforms for further performance evaluation (Step 2.2.1).

User Interface: We developed a User Interface (UI) to allow users to generate and visualize data based on their IoT application. The user interface has been developed using react JS and is responsible for: (i) registering new situations for IoT applications, (ii) enabling the transitioning between situation, and (iii) generating data based on defined situations and the configured sequence of situation transitions. The UI consists of different components for each functionality. The 'Add Situation' component (Figure 12a,b) enables to register situations and consists of several input fields: application name, context attributes, the fuzzy sets, fuzzy ranges, etc. After the application details have been added, a summarized view of the registered situations can be seen on the application dashboard (Figure 12c).

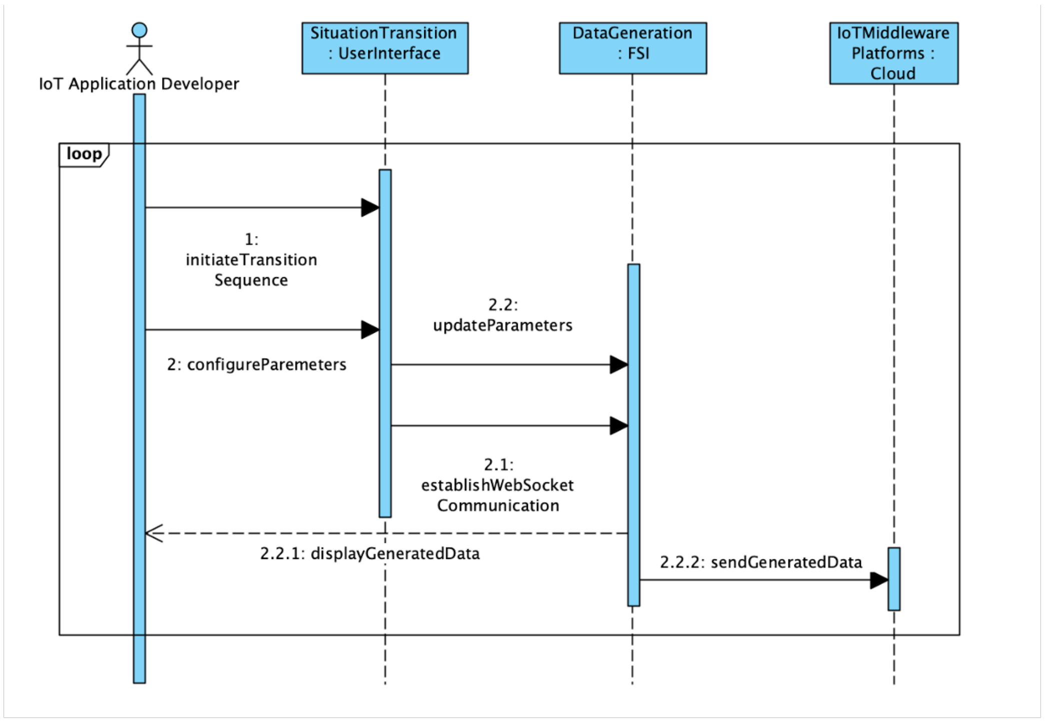
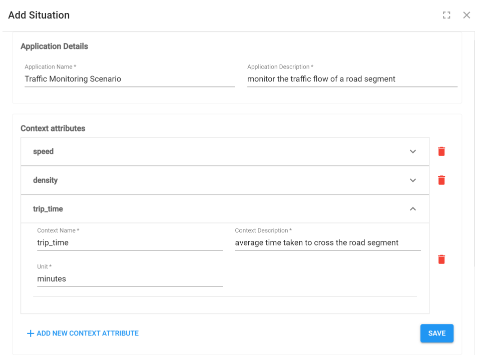

IoT Application Developer
SituationTransition : UserInterface
DataGeneration : FSI
IoTMiddleware Platforms : Cloud
loop
1: initiateTransition Sequence
2: configureParemeters
2.2: updateParameters
2.1: establishWebSocket Communication
2.2.1: displayGeneratedData
2.2.2: sendGeneratedData

Figure 11. Running situation-based IoT data generation.



Application Details

Application Name * Traffic Monitoring Scenario

Application Description * monitor the traffic flow of a road segment

Context attributes

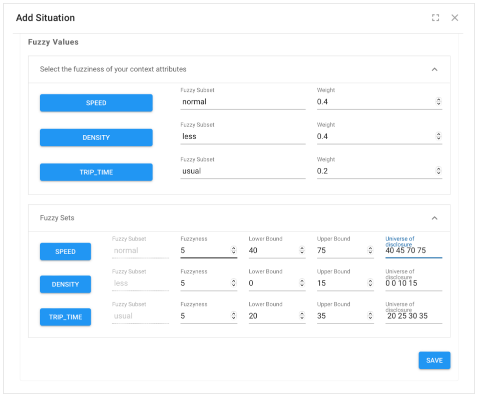
trip_time

Context Name * trip_time

Context Description * average time taken to cross the road segment

Unit * minutes

(**a**)

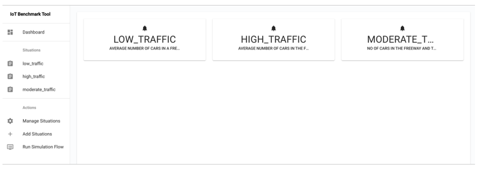


Select the fuzziness of your context attributes

	Fuzzy Subset	Weight
SPEED	normal	0.4
DENSITY	less	0.4
TRIP_TIME	usual	0.2

	Fuzzy Subset	Fuzzyness	Lower Bound	Upper Bound	Universe of disclosure
SPEED	normal	5	40	75	40 45 70 75
DENSITY	less	5	0	15	0 0 10 15
TRIP_TIME	usual	5	20	35	20 25 30 35

(**b**)



LOW_TRAFFIC
AVERAGE NUMBER OF CARS IN A FRE...

HIGH_TRAFFIC
AVERAGE NUMBER OF CARS IN THE F...

MODERATE_T...
NO OF CARS IN THE FREEWAY AND T...

(**c**)

Figure 12. Snippets of the user interface to register situations and perform simulations: (**a**) register application details, (**b**) contextual information for registering situations, (**c**) dashboard view of registered situations.

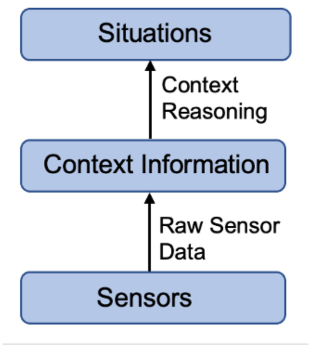
The situation transition sequence is specified by the 'Run Situation Transition' component. It has been developed using ReactJS. Redux has been used to pass data between the components as a centralized way of managing react states in the application. The required APIs for the situation repositories have been developed using NodeJS Express framework for smooth interoperability between the different components. The application can be deployed using docker for seamless integration of the services deployed in different docker container and to enable the application to run in any environment. The application is run using the docker-compose.yaml file. Additionally, Axios library has been used to send HTTP requests to the REST endpoints. The ExpressJs framework has been used to create REST API endpoints, and to request data from the database. It handles the functions to process various CRUD operations. ExpressJs uses mongoose as an Object Data Model library and connects with MongoDB service. Hence, when the request is made from ReactJS to the ExpressJS Rest API, a connection is established and the generated data are displayed to the UI console.

6. Case Study and Evaluations

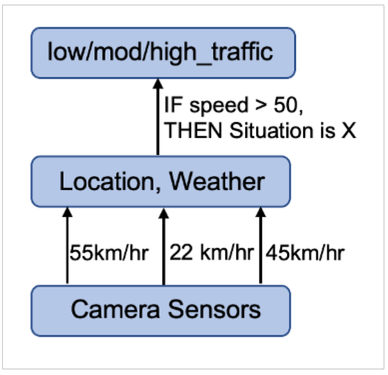
This section presents the evaluation of the proposed SA-IoTDG framework in terms of its ability to generate data based on the defined situations, enable situation transitions and evaluate performance of IoT middleware platforms. We will be using the POC prototype implementation to show the applicability of the entire framework. For that, we are using the same traffic monitoring IoT application to generate the data for the application and publish the generated data to different IoT middleware platforms to compute performance metrics. Hence, the purpose of the section is three-fold. In the rest of this section, we will first discuss a case study based evaluation to validate the proposed IoTSySML model.To this end, we have modelled a traffic monitoring IoT application based on real-world scenarios. For the second part of the evaluation, we are running in-depth evaluations by designing experiments to validate the capability of the entire framework as well as the situation transition model, which is a core part of the framework. Finally, since our aim is to use the generated data to evaluate the performance of IoT middleware platforms, we will show that our framework can be used to compare different IoT middleware platforms based on defined metrics. We will discuss the performance metrics in the subsequent sections.

6.1. Case Study: Modelling IoT Traffic Monitoring Scenario

In this section, to exemplify the proposed IoTSySML model, we apply the modelling to a traffic monitoring IoT application to validate our modelling approach. We first present a brief overview of how a real-world IoT application scenario can be modelled in terms of context theory and situations, which is illustrated using a simplified block diagram in Figure 13.



(**a**)



(**b**)

Figure 13. High-level overview of a traffic monitoring scenario: (**a**) concept, (**b**) example.

The figure depicts the concepts followed by the example of the traffic monitoring IoT application scenario. We then discuss the modelling of our case study. Figure 13a illustrates the high-level concepts: sensors send the raw sensed to the context information layer. The contextual information combined with the sensory data enables reasoning about the situations. This is instantiated in Figure 13b where data from camera sensors combined with contextual information such as current location, and weather are used to define situations such as 'low', 'moderate', or 'high' traffic scenarios.

We now discuss the modelling of the case study. With reference to IoTSySML_X, we present a SysML diagram to represent our situation aware traffic monitoring IoT application (Figure 14). We are interested in identifying the situations an entity might be involved in. For instance, our entity here is *road_x*, a road segment, which can entail situations representing the traffic flow in road_x, which can be broken down to a situation of 'low_traffic', 'moderate_traffic' or 'high_traffic'. Hence, we have a 'involved in' relationship between *road_x* and *low_traffic* situation. Situations are 'associated With' transitions, i.e., a situation of low_traffic might transition to moderate or high_traffic based on certain triggers. These have been captured by the *Transition* block in Figure 14. Situations can be associated with multiple transitions, shown by one-to-many multiplicity. The road_x entity has a *geoLocation* which represents context information. It also 'has' *CameraSensor* which 'makes' *Observations*. Figure 15 shows a JSON representation for the entity instance road_x. Camera sensors and Observations are associated with *Metadata* shown my one-to-many multiplicity in Figure 14. The observations have values 'speed', and 'density' represented by *ContextualParameters*, which are 'associated with' *Fuzzy Sets* for each observation. The fuzzy sets are in turn 'described By' FSI rules depicted by the *IF.AND.THEN* block. These rules are triggered to define the situations for our traffic scenario.

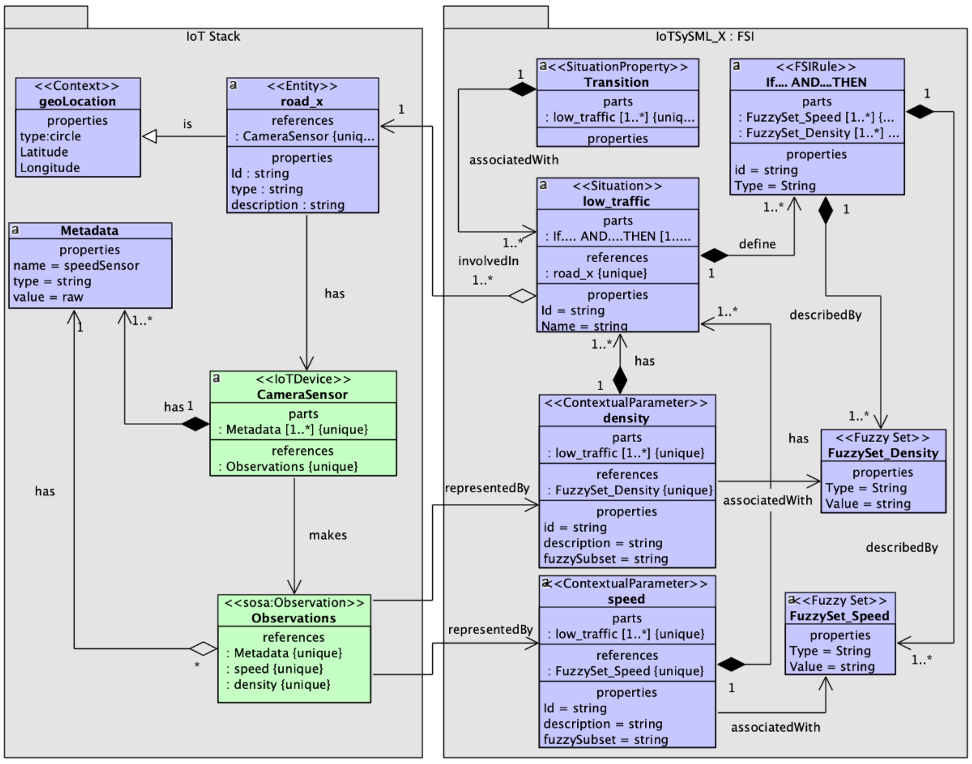
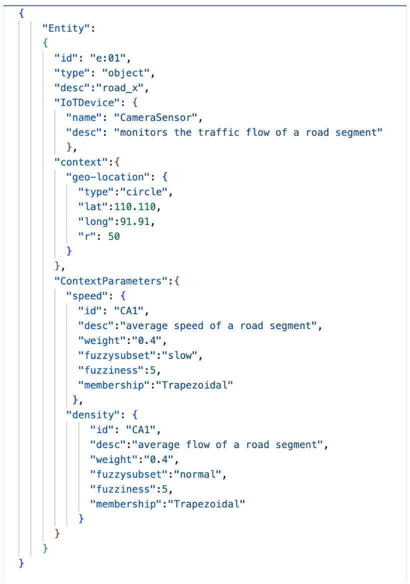


Figure 14. SysML diagram representing a Traffic Monitoring Scenario.

6.2. Experiment 1: Validating Situation Transition Model

The purpose of this section is to validate the situation transitions, which is a core component of SA-IoTDG. This experiment was conducted to evaluate the ability of the framework to support scenario based data generation. We evaluate this using our IoT traffic monitoring scenario, where we define three situations (S_1,S_2, and S_3)each allocated a probability for moving between states. For this experiment, we have assumed that the transition sequence is initiated with equally likely probabilities. Figure 16 illustrates the states involved in the transition model and the conditions which would trigger a situation to transition between the states. *'Initial state'* is the state from which we begin our transition sequence. *'Target State'* refers to the final state a situation transitions to, and *'Transition state'* is any intermediate state, a situation reaches before moving to the target state. If we exemplify the transition model with our traffic monitoring scenario, we have three pre-defined situations: 'low_traffic', 'moderate_traffic', and 'high_traffic'. We start the transition with ‘low_traffic’ and restrict the direct transition of ‘low_traffic’ to ‘high_traffic’ situation. To transition to ‘high_traffic’ situation, situations must first transition to ‘moderate_traffic’. Hence, our Initial State is ‘low_traffic’, the Target State would be ‘high_traffic’ and the Transition State would be ‘moderate_traffic’.



```
{
    "Entity":
    {
      "id": "e:01",
      "type": "object",
      "desc":"road_x",
      "IoTDevice": {
        "name": "CameraSensor",
        "desc": "monitors the traffic flow of a road segment"
        },
      "context":{
        "geo-location": {
          "type":"circle",
          "lat":110.110,
          "long":91.91,
          "r": 50
        }
      },
      "ContextParameters":{
        "speed": {
          "id": "CA1",
          "desc":"average speed of a road segment",
          "weight":"0.4",
          "fuzzysubset":"slow",
          "fuzziness":5,
          "membership":"Trapezoidal"
         },
        "density": {
            "id": "CA1",
            "desc":"average flow of a road segment",
            "weight":"0.4",
            "fuzzysubset":"normal",
            "fuzziness":5,
            "membership":"Trapezoidal"
          }
      }
    }
}
```

Figure 15. Sample JSON representation for an entity.

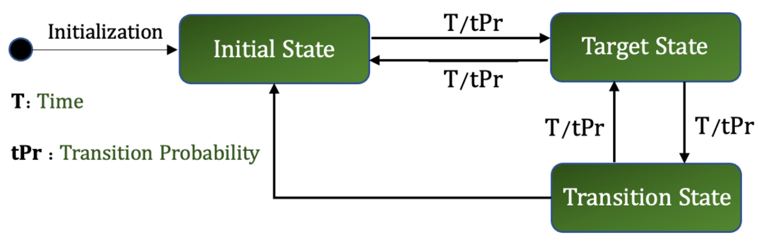


Figure 16. States and Conditions for Situation Transitions.

Situation transitions occur based on ‘time’ or based on allocated ‘Transition Probabilities (tPr)’ as illustrated in Figure 16. For a time based transition, users can specify the time during which a situation would occur, after which the next situation occurs as defined in the pre-defined sequence. For situation transitions based on transition probability, we initialize our Markov chain, define our start or initial state and allocate the probabilities for probability based transitions. We initialize our Markov model using an initial state probability vector given by:

$$P_{initial} = (p_1, p_2, p_3)$$

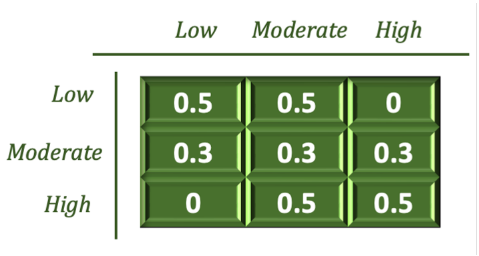
The initialization can be conducted in multiple ways: We can either assume a fixed starting point, or we can allocate probabilities for starting the Markov process. For this experiment, we assume equally likely probabilities for starting the sequence of transitions. However, this is configurable by users. Hence, they can change this depending on their application requirements. The situation transition module has been implemented in Python using the SciPy [71] library. We explain the situation transition model with the example of our traffic monitoring scenario. Since, we have three situations, ‘low_traffic’, ‘moderate_traffic’, and ‘high_traffic’, we define our initial state probability vector as:

$$p_{initial} = (0.33, 0.33, 0.33)$$

Next we also define our transition probability matrix as below:

$$\mathrm{tPr} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.3 & 0.3 & 0.3 \\ 0 & 0.5 & 0.5 \end{bmatrix}$$

Based on the above transition probability matrix tPr, and Figure 17, we can say that the situation 'low_traffic' can transition to 'low_traffic' and 'moderate_traffic' but not to 'high_traffic' since the transition probability from 'low_traffic' to 'high_traffic' is zero. Similarly, 'high_traffic' can move to 'moderate_traffic' and 'high_traffic' but not directly to 'low_traffic'. In addition, 'moderate_traffic' can transition to all the situations: 'low_traffic', 'moderate_traffic' or 'high_traffic' as illustrated by the transition probability matrix in Figure 17. Figure 18 shows our Markov model based on our transition probability matrix.



	Low	*Moderate*	*High*
Low	0.5	0.5	0
Moderate	0.3	0.3	0.3
High	0	0.5	0.5

Figure 17. Transition Probability Model of Traffic Monitoring Scenario.

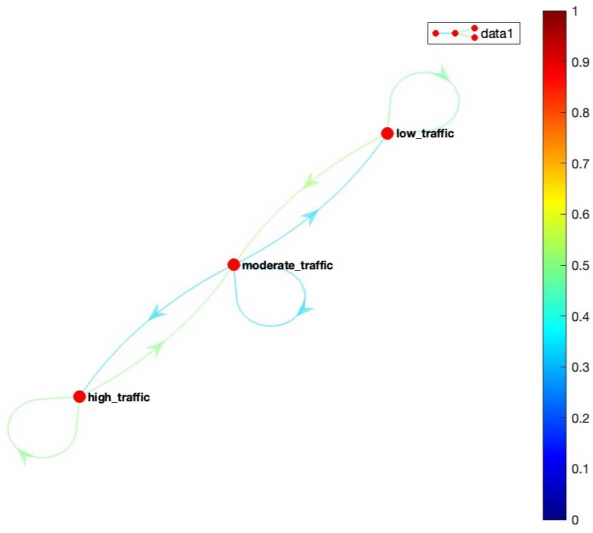


Figure 18. Markov Chain for traffic situations.

Experimental Results for Situation Transitions

We have simulated the situations of a traffic monitoring scenario. To validate our situation transition model, we have implemented a random walk based on this model in Python. We initiated our transition sequence with a situation of 'low_traffic', and the generated random walk is shown below:

$$\begin{array}{c} low\_traffic \rightarrow moderate\_traffic \rightarrow low\_traffic \rightarrow low\_traffic \rightarrow \\ moderate\_traffic \rightarrow high\_traffic \rightarrow moderate\_traffic \rightarrow moderate\_traffic \rightarrow \\ high\_traffic \rightarrow moderate\_traffic \rightarrow low\_traffic \rightarrow stop \end{array}$$

We evaluate our model by simulating a Markov sequence and visualize the result. As discussed earlier, to make the transition of situations realistic, we restricted our model to directly transition from situation 'low_traffic' to a situation of 'high_traffic' and vice-versa.

However, 'moderate_traffic' can transition to both 'low_traffic' and 'high_traffic'. This is validated in Figure 19, where the graph illustrates that there is no direct transition from situation 'low_traffic' to 'high_traffic'. The sequence starts with situation 'low_traffic', then gradually transitions to 'moderate_traffic' followed by 'high_traffic'. Hence, the model enables accurate transition from one situation to another.

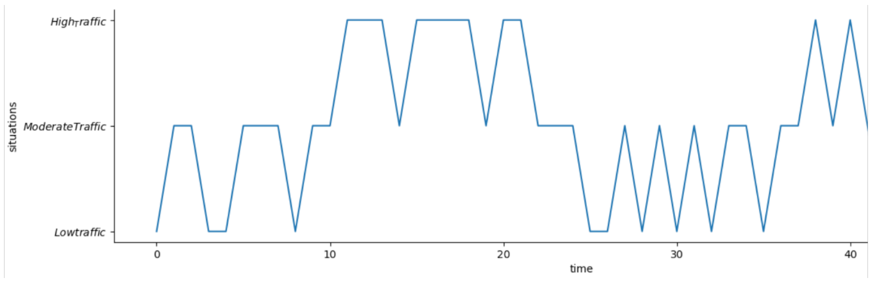


Figure 19. Markov Simulation for Situations in a Traffic Monitoring Scenario.

6.3. Experiment 2: Evaluating the Capability of the Entire Framework, SA-IoTDG

In this section, we perform a set of experiments and analysis to validate the feasibility and applicability of the proposed framework. We evaluate SA-IoTDG based on the following aspects:

- mimicking the properties of IoT time series data i.e., we aimed to investigate if SA-IoTDG is capable of producing data that have similar properties of real-world data (Experiment 2.1 and 2.2)
- capturing different user requirements and generating data dynamically (Experiment 2.3)

We performed three sets experiments to achieve the above. We will first discuss the set-up used and then discuss the designed experiments.

6.3.1. Experimental Setup and Metrics

To evaluate the proposed framework, the following set-up was used.

Open Source Tools: For this experiment, we used the IoT Data Simulator, implemented in Java, which allows both simulation and data replay. It also enables adding multiple devices and sending the device data to different target systems. We extended the tool to simulate the situations of an IoT application.

SA-IoTDG: The proposed framework for generating IoT data in line with defined situations is implemented using ReactJS, Node and Python. We have leveraged the Python FAST API framework to implement the situation description system using FSI. The situation transition module is conceptually modeled using Markov chains, and the simulations have been implemented using python and React JS. The framework is locally run using Visual Studio Code. It can be deployed using Docker as well.

IoT managed cloud platforms: The experiments were conducted on two different IoT middleware platforms, FIWARE [72] and AWS [73] to test their performance with different configurations.

We ran our experiments on a system running on MAC M1 chip, 16 GB RAM, with 1 TB SSD. During all experiments conducted, IoT data simulator were deployed using Docker. Similarly, FIWARE was also deployed as a docker container on the same system. For our experiments, we have developed a React application for simulating IoT application situations and generating relevant IoT data.

We will now discuss the metrics used to evaluate our implementations. To explore the similarities in the real and generated data by the framework, statistical parameters and autocorrelation function metrics were used.

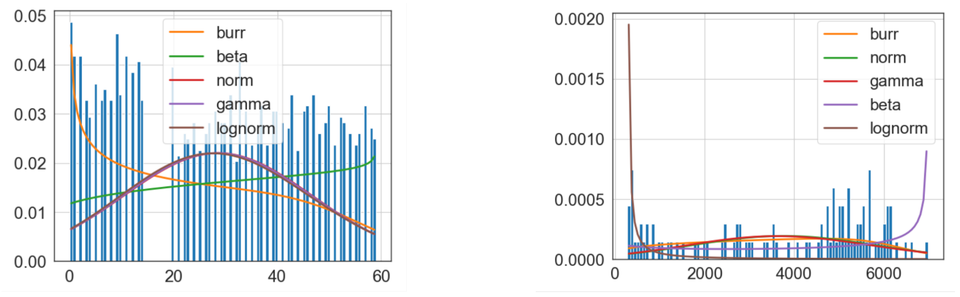
To compute the performance of IoT middleware platforms, two main metrics that were considered are: (i). data ingestion rate and (ii). query response time. We explain these metrics below:

- *Data Ingestion Delay:* The difference in the time when the data is being generated and the time at which it is being inserted into the platform.
- *Query Response Time:* By query response time, we mean the time taken to execute a query after it is triggered on detection of an event.

The next two experiments have been designed to validate the applicability of SA-IoTDG based on the two factors discussed in the beginning of this subsection.

6.3.2. Experiment 2.1: Investigating Data Distributions

To validate the data generation capability of our framework in keeping the significant features of real data, we identified the distribution of real and generated data. To achieve this, we compared the distribution of our generated traffic data with a real traffic dataset (Minnesota Department of Transportation). We found the best fitted distribution using the SciPy python library [71]. It is observed that both datasets follow a Burr distribution as the best fit based on the sumsquare_error parameter. The results are illustrated in Figure 20 which indicates that the distribution of the generated data is consistent with real-data. Hence, SA-IoTDG generates data following the same trend as real-data.

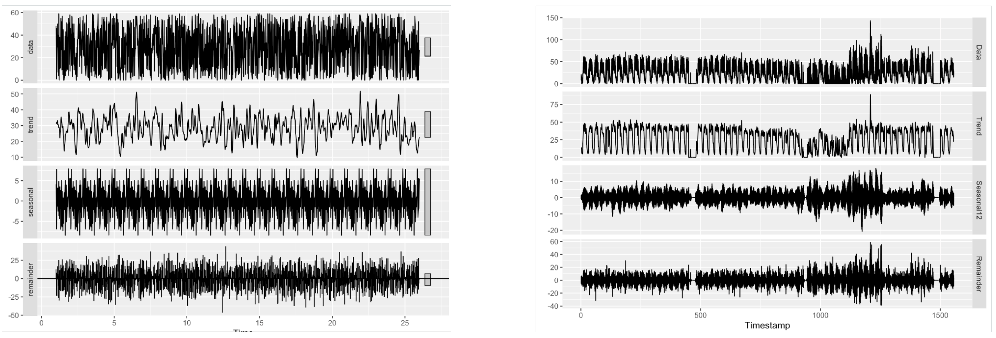


(**a**) (**b**)

Figure 20. Investigating the best fit distribution. (**a**) Distribution of generated data. (**b**) Distribution of real data.

6.3.3. Experiment 2.2: Exploring Similarity in Data Patterns

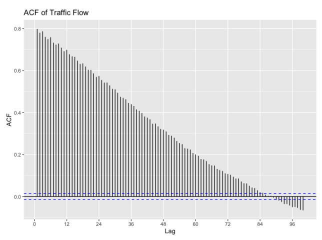
We examine the similarity between the two datasets using statistical methods. We compute statistical properties of the generated traffic data and compare with the real traffic dataset (Minnesota Department of Transportation) to evaluate if they follow similar trends and patterns. We generated high frequency time-series data with a sampling frequency of 1/5 s. We expect to observe multiple patterns in the data. Hence, we decompose our data to examine for trend and patterns. Figure 21 illustrates the components of the data. The top panels in Figure 21a,b shows the original data labelled as 'data'. The remaining panels show the existing patterns in the data. From the figure, we can infer that both datasets exhibit a similar seasonal and trend component.

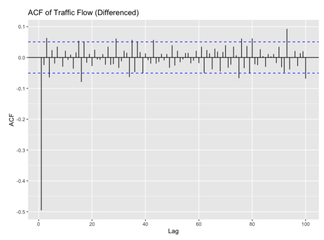


(**a**) (**b**)

Figure 21. Comparing trends and patterns in generated and real data. (**a**) Components of generated data. (**b**) Components of real data.

Since the generated data exhibits a trend component, we can say that the data are non-stationary. We confirm the non stationarity of the generated traffic data using autocorrelation function (ACF). ACF gives the correlation between the present values of a time-series data and the past values. The gradual decrease of the ACF in Figure 22a illustrates that the generated data are non-stationary. In order to fit the generated data in any model for further analysis, we need stationary data, so we transform the data to stationary by differencing them. Figure 22b shows the differenced data.





(**a**) (**b**)

Figure 22. Autocorrelation functions of generated data. (**a**) ACF of generated traffic Data. (**b**) ACF of differenced data.

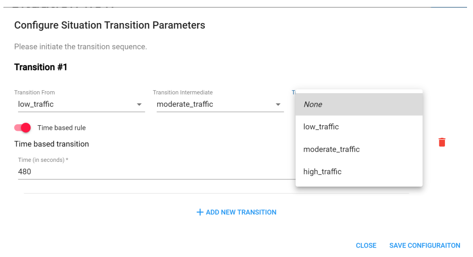
Additionally, we compute a statistical summary presented in Table 5. It can be observed that both datasets have very similar statistical properties, which can be used to infer that generated data follows properties similar to that of a real dataset.

Table 5. Comparing statistical summary of datasets.

Input Variables	Generated Data	Real Data
Mean	0.00	0.00
SD	18.34	17.34
1st Quartile	10.00	11.00
Median	29	30
3rd Quartile	40	44

6.3.4. Experiment 2.3: Data Generation with Varying User Configuration

These experiments were designed to validated that SA-IoTDG is capable of capturing user specific requirements and generate data based on the specified configuration. We instantiate this using two examples. (1). Generating data by specifying a threshold time, and (2). specifying the data generation frequency in seconds. Figure 23a illustrates a configuration specified for 8 min (480 s) of data to be generated for each situation. We have modelled a traffic monitoring application as discussed, entailing situations of low_traffic, moderate_traffic, and high_traffic. Figure 23b illustrates data being generated for 8 min for each traffic situation.


Configure Situation Transition Parameters

Please initiate the transition sequence.

Transition #1

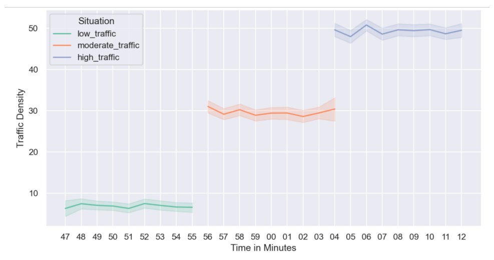
Transition From: low_traffic

Transition Intermediate: moderate_traffic

Time based rule

Time based transition

Time (in seconds) *: 480



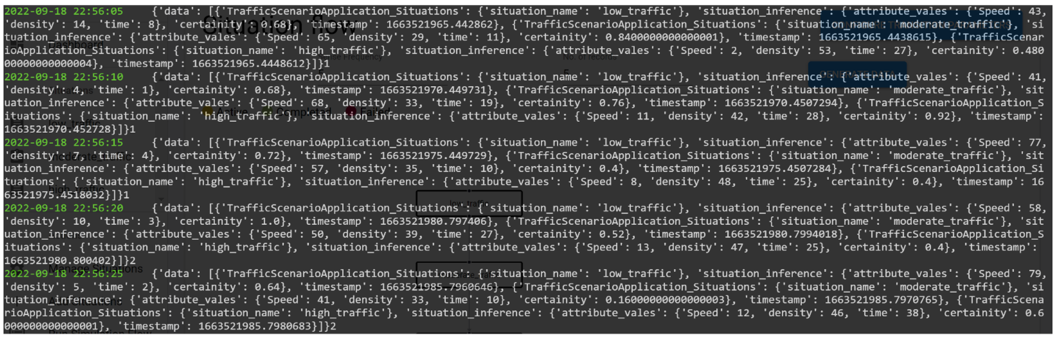
(**a**) (**b**)

Figure 23. (**a**) Specifyingtime configuration. (**b**) Generated data based on configured time.

Figure 24 illustrates a snapshot of data being generated by SA-IoTDG at a user configured frequency of 5 s. We also simulated data, mimicking a traffic monitoring application scenario. The data was aggregated at hour level. Figure 25 shows a sample of simulated data with the peaks' representation data for a situation of high_traffic.

6.4. Experiment 3: Using SA-IoTDG to Conduct Performance Evaluation of IoT Middleware Platforms

This experiment was conducted to validate the frameworks' capability to evaluate the performance of different IoT middleware platforms. To this end, we used the same traffic monitoring IoT application to generate the data for the application and published the generated data to two IoT middleware platforms AWS and FIWARE. The performance was analyzed based on metrics data ingestion delay and query response time as discussed in Section 6.3.1. Data ingestion delay was calculated by publishing the generated data to the two middleware platforms with a data ingestion rate (rate at which sensor data being inserted to a system per second) of 500 data points per second. Figure 26a compares the data ingestion delay of AWS and FIWARE. The result shows that the data ingestion delay is slightly more in FIWARE than that of AWS. Other metrics such as the query response time can also be computed by querying the ingested data on both platforms. Figure 26b illustrates how query response time varies with different data ingestion rates and varying complexity of queries. The experiment was conducted with data ingestion rates of 200, 500 and 1000 data per second, while varying the query complexity at the same time. The query complexity was varied by changing the number of joins and filter conditions. Since, the focus of the paper is not on evaluating the performance of the IoT middleware platforms, we did not provide a detailed technical evaluation. The aim is rather to show that the proposed framework can be used to conduct such performance evaluations.



```
2022-09-18 22:56:05    {'data': [{'TrafficScenarioApplication_Situations': {'situation_name': 'low_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 43,
'density': 14, 'time': 8}, 'certainity': 0.68}, 'timestamp': 1663521965.442862}, {'TrafficScenarioApplication_Situations': {'situation_name': 'moderate_traffic'}, 'si
tuation_inference': {'attribute_vales': {'Speed': 65, 'density': 29, 'time': 11}, 'certainity': 0.8400000000000001}, 'timestamp': 1663521965.4438615}, {'TrafficScenar
ioApplication_Situations': {'situation_name': 'high_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 2, 'density': 53, 'time': 27}, 'certainity': 0.480
00000000000004}, 'timestamp': 1663521965.4448612}]}1
2022-09-18 22:56:10    {'data': [{'TrafficScenarioApplication_Situations': {'situation_name': 'low_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 41,
'density': 4, 'time': 1}, 'certainity': 0.68}, 'timestamp': 1663521970.449731}, {'TrafficScenarioApplication_Situations': {'situation_name': 'moderate_traffic'}, 'sit
uation_inference': {'attribute_vales': {'Speed': 68, 'density': 33, 'time': 19}, 'certainity': 0.76}, 'timestamp': 1663521970.4507294}, {'TrafficScenarioApplication_S
ituations': {'situation_name': 'high_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 11, 'density': 42, 'time': 28}, 'certainity': 0.92}, 'timestamp':
1663521970.452728}]}1
2022-09-18 22:56:15    {'data': [{'TrafficScenarioApplication_Situations': {'situation_name': 'low_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 77,
'density': 7, 'time': 4}, 'certainity': 0.72}, 'timestamp': 1663521975.449729}, {'TrafficScenarioApplication_Situations': {'situation_name': 'moderate_traffic'}, 'sit
uation_inference': {'attribute_vales': {'Speed': 57, 'density': 35, 'time': 10}, 'certainity': 0.4}, 'timestamp': 1663521975.4507284}, {'TrafficScenarioApplication_Si
tuations': {'situation_name': 'high_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 8, 'density': 48, 'time': 25}, 'certainity': 0.4}, 'timestamp': 16
63521975.4518032}]}1
2022-09-18 22:56:20    {'data': [{'TrafficScenarioApplication_Situations': {'situation_name': 'low_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 58,
'density': 10, 'time': 3}, 'certainity': 1.0}, 'timestamp': 1663521980.797406}, {'TrafficScenarioApplication_Situations': {'situation_name': 'moderate_traffic'}, 'sit
uation_inference': {'attribute_vales': {'Speed': 50, 'density': 39, 'time': 27}, 'certainity': 0.52}, 'timestamp': 1663521980.7994018}, {'TrafficScenarioApplication_S
ituations': {'situation_name': 'high_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 13, 'density': 47, 'time': 25}, 'certainity': 0.4}, 'timestamp':
1663521980.800402}]}2
2022-09-18 22:56:25    {'data': [{'TrafficScenarioApplication_Situations': {'situation_name': 'low_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 79,
'density': 5, 'time': 2}, 'certainity': 0.64}, 'timestamp': 1663521985.7960646}, {'TrafficScenarioApplication_Situations': {'situation_name': 'moderate_traffic'}, 'si
tuation_inference': {'attribute_vales': {'Speed': 41, 'density': 33, 'time': 10}, 'certainity': 0.16000000000000003}, 'timestamp': 1663521985.7970765}, {'TrafficScena
rioApplication_Situations': {'situation_name': 'high_traffic'}, 'situation_inference': {'attribute_vales': {'Speed': 12, 'density': 46, 'time': 38}, 'certainity': 0.6
000000000000001}, 'timestamp': 1663521985.7980683}]}2
```

Figure 24. Data generated every 5 s.

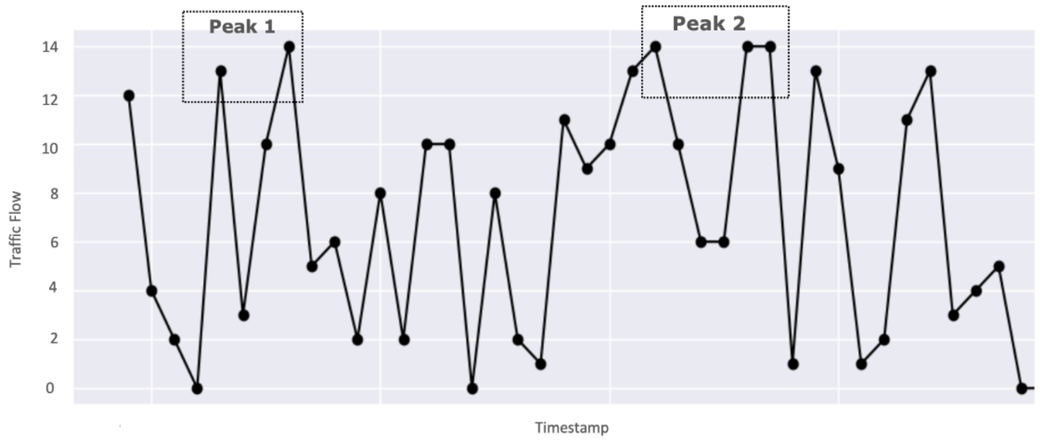


Figure 25. Visualizing generated data of a traffic monitoring scenario.

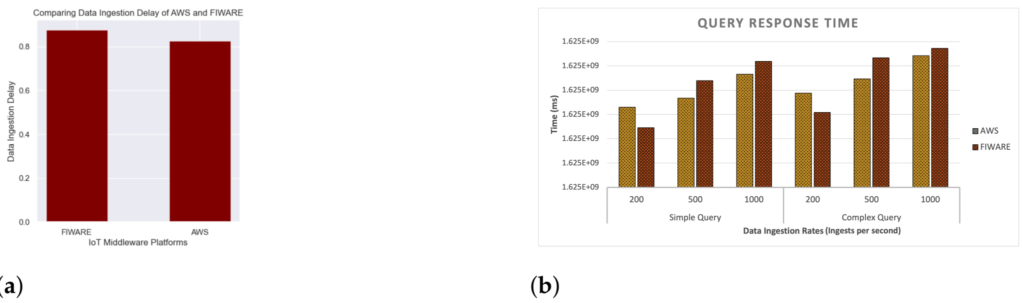


Figure 26. (**a**) Data ingestion delay. (**b**) Query response time of AWS and FIWARE [15].

7. Conclusions

In this paper, we proposed, implemented and evaluated SA-IoTDG, a novel framework to capture and simulate situation based IoT different application requirements and generate data. The proposed framework can: (1) define rules that can simulate real-world situations and stipulate the transitions of the defined situations, (2) generate IoT data based on the user specified configurations, and (3) use the framework to conduct performance evaluations of IoT middleware platforms. In existing works, we can currently generate data for single situations. The advantage of our proposed method is that, with our proposal, we can model multiple situations, specify the transition sequence and generate data. To the best of our knowledge, this is the first work to consider the transition of situations for generating IoT data. We illustrated that the proposed SA-IoTDG is capable of working with different configurations, and generates data with features similar to that of real data. Hence, SA-IoTDG is configurable and dynamic and is a step towards accelerating the development of a benchmark solution for IoT middleware platforms.

In this paper, our main focus was on generating IoT data relevant to situation specific IoT applications for evaluating IoT middleware platforms. Despite the contribution, there are some open issues that require further investigation. One of the interesting aspects we are planning to work on is:

Data Extrapolation: By data extrapolation, we mean generating data based on a sample data. In this paper, we have conducted experiments, where data was generated based on pre-defined rules by users. However, to provide more flexibility to users, and generate more realistic data, the framework should be able to generate data based on a sample data. To address this requirement, we plan to investigate and develop a mechanism to provide a dataset as seed and generate similar data for relevant IoT applications. Additionally, considering this work as a starting point, as future work, we plan to conduct further evaluations considering additional metrics to evaluate the performance of IoT middlewares.

Author Contributions: Conceptualization, S.M., A.H., P.P.J., P.D.H. and D.G.; methodology, S.M., P.D.H., P.P.J. and A.H.; writing—original draft preparation, S.M., P.D.H., P.P.J. and A.H.; writing—review and editing, S.M., P.D.H., P.P.J. and A.H.; supervision, P.P.J., P.D.H., A.H. and D.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. News, I.B. The IoT in 2030: 24 Billion Connected Things Generating 1.5 Trillion Dollar. 2020. Available online: https://iotbusinessnews.com/2020/05/20/03177-the-iot-in-2030-24-billion-connected-things-generating-1-5-trillion (accessed on 1 November 2020).
2. Han, S.N.; Lee, G.M.; Crespi, N.; Heo, K.; Van Luong, N.; Brut, M.; Gatellier, P. DPWSim: A simulation toolkit for IoT applications using devices profile for web services. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Republic of Korea, 6–8 March 2014; pp. 544–547.
3. TPC-H Benchmark. 2020. Available online: http://www.tpc.org/tpch/ (accessed on 1 November 2020).
4. TPC Benchmark DS (TPC-DS). 2020. Available online: http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf (accessed on 1 November 2020).
5. Ghazal, A.; Ivanov, T.; Kostamaa, P.; Crolotte, A.; Voong, R.; Al-Kateb, M.; Ghazal, W.; Zicari, R.V. Bigbench v2: The new and improved bigbench. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017 ; pp. 1225–1236.
6. Zhao, J.M.; Wang, W.S.; Liu, X.; Chen, Y.F. Big data benchmark-big DS. In *Advancing Big Data Benchmarks*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 49–57.
7. Wang, L.; Zhan, J.; Luo, C.; Zhu, Y.; Yang, Q.; He, Y.; Gao, W.; Jia, Z.; Shi, Y.; Zhang, S.; et al. Bigdatabench: A big data benchmark suite from internet services. In Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, USA, 15–19 February 2014 ; pp. 488–499.
8. Shukla, A.; Chaturvedi, S.; Simmhan, Y. RIoTBench: An iot benchmark for distributed stream processing systems. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4257. [CrossRef]
9. Kolbe, N.; Zaslavsky, A.; Kubler, S.; Robert, J.; Traon, Y.L. Enriching a situation awareness framework for IoT with knowledge base and reasoning components. In Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context, Trento, Italy, 20–22 November 2017; pp. 41–54.
10. Padovitz, A.; Loke, S.W.; Zaslavsky, A. Towards a theory of Context Spaces. In Proceedings of the IEEE Annual Conference on Pervasive Computing and Communications Workshops, Orlando, FL, USA, 12–17 March 2004; pp. 38–42.
11. Poess, M.; Nambiar, R.; Kulkarni, K.; Narasimhadevara, C.; Rabl, T.; Jacobsen, H.A. Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 1519–1530.
12. Cooper, B.F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing, Indianapolis, IN, USA, 10–11 June 2010; pp. 143–154.
13. Liu, X.; Golab, L.; Golab, W.M.; Ilyas, I.F. Benchmarking Smart Meter Data Analytics. In Proceedings of the EDBT, Brussels, Belgium, 23–27 March 2015; pp. 385–396.
14. Iftikhar, N.; Liu, X.; Nordbjerg, F.E.; Danalachi, S. A prediction-based smart meter data generator. In Proceedings of the 2016 19th International Conference on Network-Based Information Systems (NBiS), Ostrava, Czech Republic, 7–9 September 2016; pp. 173–180.
15. Mondal, S.; Hassani, A.; Jayaraman, P.P.; Delir Haghighi, P.; Georgakopoulos, D. Modelling IoT Application Requirements for Benchmarking IoT Middleware Platforms. In Proceedings of the 23rd International Conference on Information Integration and Web Intelligence, Linz, Austria, 29 November–1 December 2021; pp. 553–561.
16. Delir Haghighi, P.; Krishnaswamy, S.; Zaslavsky, A.; Gaber, M.M. Reasoning about context in uncertain pervasive computing environments. In Proceedings of the European Conference on Smart Sensing and Context, Zurich, Switzerland, 29–31 October 2008; pp. 112–125.
17. Cardoso, J.; Pereira, C.; Aguiar, A.; Morla, R. Benchmarking IoT middleware platforms. In Proceedings of the 2017 IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China, 12–15 June 2017; pp. 1–7.
18. Pereira, C.; Cardoso, J.; Aguiar, A.; Morla, R. Benchmarking Pub/Sub IoT middleware platforms for smart services. *J. Reliab. Intell. Environ.* **2018**, *4*, 25–37. [CrossRef]
19. Scott, R.; Östberg, D. A Comparative Study of Open-Source IoT Middleware Platforms. 2018. Available online: https://www.semanticscholar.org/paper/A-comparative-study-of-open-source-IoT-middleware-Scott-Ostbergfbaf7491ea0acbc44560d173b7287366799fb586 (accessed on 1 June 2021).
20. Agarwal, P.; Alam, M. Investigating IoT middleware platforms for smart application development. In *Smart Cities—Opportunities and Challenges*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 231–244.
21. Medvedev, A.; Hassani, A.; Zaslavsky, A.; Jayaraman, P.P.; Indrawan-Santiago, M.; Haghighi, P.D.; Ling, S. Data ingestion and storage performance of iot platforms: Study of openiot. In Proceedings of the International Workshop on Interoperability and Open-Source Solutions, Stuttgart, Germany, 7 November 2016; pp. 141–157.
22. Medvedev, A.; Hassani, A.; Zaslavsky, A.; Haghighi, P.D.; Ling, S.; Jayaraman, P.P. Benchmarking IoT context management platforms: High-level queries matter. In Proceedings of the 2019 Global IoT Summit (GIoTS), Aarhus, Denmark, 17–21 June 2019; pp. 1–6.
23. Salhofer, P. Evaluating the FIWARE Platform. In Proceedings of the 51st Hawaii International Conference on System Sciences, Hilton Waikoloa Village, Hawaii, HI, USA, 3–6 January 2018.

24. da Cruz, M.A.; Rodrigues, J.J.; Sangaiah, A.K.; Al-Muhtadi, J.; Korotaev, V. Performance evaluation of IoT middleware. *J. Netw. Comput. Appl.* **2018**, *109*, 53–65. [CrossRef]
25. Araujo, V.; Mitra, K.; Saguna, S.; Åhlund, C. Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities. *J. Parallel Distrib. Comput.* **2019**, *132*, 250–261. [CrossRef]
26. Martínez, R.; Pastor, J.Á.; Álvarez, B.; Iborra, A. A testbed to evaluate the fiware-based IoT platform in the domain of precision agriculture. *Sensors* **2016**, *16*, 1979. [CrossRef]
27. Lee, C.I.; Lin, M.Y.; Yang, C.L.; Chen, Y.K. IoTBench: A benchmark suite for intelligent Internet of Things edge devices. In Proceedings of the 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 22–25 September 2019; pp. 170–174.
28. TPCx-IoT. 2021. Available online: http://tpc.org/tpcx-iot/default5.asp (accessed on 1 June 2021).
29. Bahga, A.; Madisetti, V.K. Synthetic workload generation for cloud computing applications. *J. Softw. Eng. Appl.* **2011**, *4*, 396. [CrossRef]
30. TPCx-HS Benchmark. Available online: https://www.tpc.org/tpcx-hs/(accessed on 1 October 2021).
31. Huang, S.; Huang, J.; Dai, J.; Xie, T.; Huang, B. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In Proceedings of the 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW), Long Beach, CA, USA, 6–10 March 2010; pp. 41–51.
32. Agarwal, P.; Alam, M. Investigating IoT Middleware Platforms for Smart Application Development. *arXiv* **2018**, arXiv:1810.12292.
33. Ali, M.I.; Gao, F.; Mileo, A. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In Proceedings of the International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; pp. 374–389.
34. Le-Phuoc, D.; Dao-Tran, M.; Pham, M.D.; Boncz, P.; Eiter, T.; Fink, M. Linked stream data processing engines: Facts and figures. In Proceedings of the International Semantic Web Conference, Boston, MA, USA, 11–15 November 2012; pp. 300–312.
35. Almakhdhub, N.S.; Clements, A.A.; Payer, M.; Bagchi, S. Benchiot: A security benchmark for the internet of things. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, OR, USA, 24–27 June 2019; pp. 234–246.
36. Arlitt, M.; Marwah, M.; Bellala, G.; Shah, A.; Healey, J.; Vandiver, B. Iotabench: An internet of things analytics benchmark. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, 28 January–4 February 2015; pp. 133–144.
37. Arasu, A.; Cherniack, M.; Galvez, E.; Maier, D.; Maskey, A.S.; Ryvkina, E.; Stonebraker, M.; Tibbetts, R. Linear road: A stream data management benchmark. In Proceedings of the Thirtieth International Conference on Very Large Data, Toronto, ON, Canada, 31 August–3 September 2004; Volume 30, pp. 480–491.
38. Zeng, X.; Garg, S.K.; Strazdins, P.; Jayaraman, P.P.; Georgakopoulos, D.; Ranjan, R. IOTSim: A simulator for analysing IoT applications. *J. Syst. Archit.* **2017** , *72*, 93–107. [CrossRef]
39. Alsaedi, A.; Moustafa, N.; Tari, Z.; Mahmood, A.; Anwar, A. TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access* **2020**, *8*, 165130–165150. [CrossRef]
40. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
41. Gupta, H.; Vahid Dastjerdi, A.; Ghosh, S.K.; Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* **2017**, *47*, 1275–1296. [CrossRef]
42. Nazir, B.; Hasbullah, H. Energy balanced clustering in wireless sensor network. In Proceedings of the 2010 International Symposium on Information Technology, Kuala Lumpur, Malaysia, 15–17 June 2010; Volume 2, pp. 569–574.
43. Roussel, K.; Song, Y.Q.; Zendra, O. Using Cooja for WSN Simulations: Some New Uses and Limits. 2016. Available online: https://dl.acm.org/doi/10.5555/2893711.2893790 (accessed on 1 June 2021).
44. Issariyakul, T.; Hossain, E. Introduction to network simulator 2 (NS2). In *Introduction to Network Simulator NS2*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–18.
45. Li, X.S.; Tao, X.P.; Song, W.; Dong, K. Aocml: A domain-specific language for model-driven development of activity-oriented context-aware applications. *J. Comput. Sci. Technol.* **2018**, *33*, 900–917. [CrossRef]
46. Henricksen, K.; Indulska, J. Modelling and using imperfect context information. In Proceedings of the IEEE Annual Conference on Pervasive Computing and Communications Workshops, Orlando, FL, USA, 12–17 March 2004; pp. 33–37.
47. Boytsov, A.; Zaslavsky, A.; Synnes, K. Extending Context Spaces theory by predicting run-time context. In *Smart Spaces and Next Generation Wired/Wireless Networking*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 8–21.
48. Padovitz, A.; Zaslavsky, A.; Loke, S.W. A Unifying Model for Representing and Reasoning about Context under Uncertainty. 2006. Available online: http://arrow.latrobe.edu.au:8080/vital/access/manager/Repository/latrobe:2524?f0=sm_creator%3A%22Padovitz%2C+Amir.%22 (accessed on 1 June 2021).
49. Xi, Z.; Panoutsos, G. Interpretable machine learning: Convolutional neural networks with RBF Fuzzy Logic classification rules. In Proceedings of the 2018 International Conference on Intelligent Systems (IS), Phuket, Thailand, 17–19 November 2018; pp. 448–454.
50. Chiesa, G.; Di Vita, D.; Ghadirzadeh, A.; Herrera, A.H.M.; Rodriguez, J.C.L. A fuzzy-logic IoT lighting and shading control system for smart buildings. *Autom. Constr.* **2020**, *120*, 103397. [CrossRef]

51. Haghighi, P.D.; Krishnaswamy, S.; Zaslavsky, A.; Gaber, M.M.; Sinha, A.; Gillick, B. Open mobile miner: A toolkit for building situation-aware data mining applications. *J. Organ. Comput. Electron. Commer.* **2013**, *23*, 224–248. [CrossRef]
52. Klappenecker, A.; Lee, H.; Welch, J.L. Finding available parking spaces made easy. In Proceedings of the 6th International Workshop on Foundations of Mobile Computing, Cambridge, MA, USA, 16 September 2010; pp. 49–52.
53. Blanchet, J.; Gallego, G.; Goyal, V. A Markov chain approximation to choice modeling. *Oper. Res.* **2016**, *64*, 886–905. [CrossRef]
54. Li, N.; Kolmanovsky, I.; Girard, A.; Filev, D. Fuzzy encoded Markov chains: Overview, observer theory, and applications. *IEEE Trans. Syst. Man, Cybern. Syst.* **2020**, *51*, 116–130. [CrossRef]
55. Li, G.; Xiu, B. Fuzzy Markov chains based on the fuzzy transition probability. In Proceedings of the 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 31 May–2 June 2014; pp. 4351–4356.
56. Filev, D.P.; Kolmanovsky, I. Generalized Markov models for real-time modeling of continuous systems. *IEEE Trans. Fuzzy Syst.* **2013**, *22*, 983–998. [CrossRef]
57. IoT Data Simulator. 2020. Available online: https://github.com/IBA-Group-IT/IoT-data-simulator (accessed on 15 October 2022).
58. Boytsov, A.; Zaslavsky, A. From sensory data to situation awareness: Enhanced Context Spaces theory approach. In Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, Washington, DC, USA, 12–14 December 2011; pp. 207–214.
59. Basu, A.K. *Introduction to Stochastic Processes*; Alpha Science Int'l Ltd.: Oxford, UK, 2003.
60. W3Cs. Semantic Sensor Network Ontology. 2019. Available online: https://www.w3.org/TR/vocab-ssn (accessed on 18 March 2021).
61. W3C. IoT-Lite Ontology. 2021. Available online: https://www.w3.org/Submission/iot-lite (accessed on 18 March 2021).
62. Patroumpas, K.; Sellis, T. Event processing and real-time monitoring over streaming traffic data. In Proceedings of the International Symposium on Web and Wireless Geographical Information Systems, Naples, Italy, 12–13 April 2012; pp. 116–133.
63. Târnaucă, B.; Puiu, D.; Damian, D.; Comnac, V. Traffic condition monitoring using complex event processing. In Proceedings of the 2013 International Conference on System Science and Engineering (ICSSE), Budapest, Hungary, 4–6 July 2013; pp. 123–128.
64. Cárdenas-Benítez, N.; Aquino-Santos, R.; Maga na-Espinoza, P.; Aguilar-Velazco, J.; Edwards-Block, A.; Medina Cass, A. Traffic congestion detection system through connected vehicles and big data. *Sensors* **2016**, *16*, 599. [CrossRef]
65. Schömig, N.; Metz, B. Three levels of situation awareness in driving with secondary tasks. *Saf. Sci.* **2013**, *56*, 44–51. [CrossRef]
66. Mamdani, E.H. Advances in the linguistic synthesis of fuzzy controllers. *Int. J. Man-Mach. Stud.* **1976**, *8*, 669–678. [CrossRef]
67. Norris, J.R.; Norris, J.R. *Markov Chains*; Number 2; Cambridge University Press: Cambridge, UK, 1998.
68. Kamilaris, A.; Gao, F.; Prenafeta-Boldu, F.X.; Ali, M.I. Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications. In Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016; pp. 442–447.
69. Compton, M.; Barnaghi, P.; Bermudez, L.; Garcia-Castro, R.; Corcho, O.; Cox, S.; Graybeal, J.; Hauswirth, M.; Henson, C.; Herzog, A.; et al. The SSN ontology of the W3C semantic sensor network incubator group. *J. Web Semant.* **2012**, *17*, 25–32. [CrossRef]
70. Janowicz, K.; Haller, A.; Cox, S.J.; Le Phuoc, D.; Lefrançois, M. SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *J. Web Semant.* **2019**, *56*, 1–10. [CrossRef]
71. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef] [PubMed]
72. Fiware. Available online: https://www.fiware.org (accessed on 31 July 2020).
73. Amazon Web Service IoT. Available online: https://aws.amazon.com/iot (accessed on 31 July 2020).