Semantic Model-Driven Framework for Validating Quality Requirements of Internet of Things Streaming Data

by

OLUWASEUN O. BAMGBOYE

Thesis submitted in partial fulfilment of the requirements

of Edinburgh Napier University, for the award of

Doctor of Philosophy

School of Computing

Edinburgh Napier University

JULY 2021

Dedication

In memory of my father. Late E.O. Bamgboye. You left fingerprints of selflessness and hard work on my life. You shall not be forgotten...

Author's declaration

I, Oluwaseun Bamgboye, declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.

Where I have consulted the published work of others this is always clearly attributed.

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work.

I have acknowledged all main sources of help.

If my research follows on from previous work or is part of a larger collaborative research project, I have made clear exactly what was done by others and what I have contributed myself.

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

SIGNED: ..... DATE: .....

Acknowledgements

For unto God be all glory and majesty, who in his infinite mercies has made me to be a witness unto this day.
I would like to start by showing my profound gratitude to my Director of Studies, Prof. Xiaodong Liu for his immeasurable supports throughout my PhD study. I appreciate the continuous professional guidance both academically and career wise. I am indeed lucky to have been supervised by him. My appreciation also goes to my Supervisor, Peter Cruickshank for his support and guidance through critical thinking during the doctorate degree. I also want to thank my Panel Chair, Dr. Michael Smyth for his pastoral role by making sure my mental and emotional needs are always in the balance.

To more important people in my life, I would like to thank my family, especially my wife, Oladunni and my wonderful children (Oreofe, Semilore and Florence) who have shown their supports and understanding when I have been busy during the period of my study. Special thanks to my wonderful mother for the word of encouragements when the roads are rough, and also to my siblings (Sola, Tolu, Yomi, Bola, Lara, Dele)that kept checking on my welfare, especially during the most challenging times. I cannot forget to mention my distant uncle Mr R. Farombi for all his prayers towards the completion of the program. Also to a friend like a little brother to me, Adedeji Oyeleye for all his technical supports on getting around the programming frameworks.

Much thanks to the funding bodies of my PhD programme. In particular, I appreciate Tetfund under the leadership of Federal government of Nigeria (with support from Moshood Abiola Polytechnic, Ojere), and the school of computing, Edinburgh Napier University, who graciously supported the funding of my PhD study when I almost became stranded on the program.

Abstract

The rise of Internet of Things has provided platforms mostly enhanced by real-time data-driven services for reactive services and Smart Cities innovations. However, IoT streaming data are known to be compromised by quality problems, thereby influencing the performance and accuracy of IoT-based reactive services or Smart applications. This research investigates the suitability of the semantic approach for the run-time validation of IoT streaming data for quality problems. To realise this aim, Semantic IoT Streaming Data Validation with its framework (SISDaV) is proposed. The novel approach involves technologies for semantic query and reasoning with semantic rules defined on an established relationship with external data sources with consideration for specific run-time events that can influence the quality of streams. The work specifically targets quality issues relating to inconsistency, plausibility, and incompleteness in IoT streaming data.

In particular, the investigation covers various RDF stream processing and rule-based reasoning techniques and effects of RDF Serialised formats on the reasoning process. The contributions of the work include the hierarchy of IoT data stream quality problem, lightweight evolving Smart Space and Sensor Measurement Ontology, generic time-aware validation rules and, SISDaV framework- a unified semantic rule-based validation system for RDF-based IoT streaming data that combines the popular RDF stream processing the system with generic enhanced time-aware rules.

The semantic validation process ensures the conformance of the raw streaming data value produced by the IoT node(s) with IoT streaming data quality requirements and the expected value. This is facilitated through a set of generic continuous validation rules, which has been realised by extending the popular Jena rule syntax with a time element. The comparative evaluation of SISDaV is based on its effectiveness and efficiency based on the expressivity of the different serialised RDF data formats.

The results are interpreted with relevant statistical estimations and performance metrics. The results from the evaluation approve of the feasibility of the framework in terms of containing the semantic validation process within the interval between reads of sensor nodes as well as provision of additional requirements that can enhance IoT streaming data processing systems which are currently missing in most related state-of-art RDF stream processing systems. Furthermore, the approach can satisfy the main research objectives as identified by the study.

Abbreviation

- **CEP:** Complex Event Processing
- **DSMS:** Data Stream Management System
- **EEG:** Electroencephalogram
- **GSN:** Global Sensor Network
- **IoT:** Internet of Things
- **IRI:** Internationalise Resource Identifier
- **JSON:** JavaScript Object Notation
- **OWL:** Web Ontology Language
- **RDF:** Resource Description Framework
- **RDFS:** RDF Schema
- **RSP:** RDF Stream Processing
- **SPARQL:** Simple Protocol And RDF Query Language
- **SSN:** Semantic Sensor Network
- **URI:** Uniform Resource Identifier
- **W3C:** World Wide Web Consortium

TABLE OF CONTENTS

LIST OF TABLES

TABLES **Page**

LIST OF FIGURES

FIGURES **Page**

First Chapter

Introduction

1.1 Problem Statement

The Internet of Things (IoT) focuses on infrastructure issues by identifying and connecting real-life objects. In this domain, the streaming data produced from these objects are useful in driving actuation and data-driven services on the web with support of semantic stream processing systems. Despite growing efforts to apply the semantic stream processing approaches to provide a unifying model, foundations, techniques, and tools necessary to integrate data streams with semantic processing systems, the only focus has been to improve interoperability through semantic stream querying (Anicic et al., 2011; Barbieri et al., 2009; Calbimonte et al., 2010; Le-Phuoc et al., 2011). There still exists significant research gaps in the application of semantic stream processing systems in terms of managing the quality requirements of IoT data streams (Karkouch et al., 2016), providing reasoning capabilities and, integrating background knowledge with streams(Valle et al., 2009). As a result, this calls for an approach to manage these issues with consideration for IoT streaming data towards the realisation of both effective and efficient data-driven service delivery at the lowest granularity levels of IoT applications. Furthermore, there is need to consider the direct impact of the quality issues attributed to IoT streaming data on data-driven decision-support systems at the operational level of smart city. This implies that a poor quality IoT streaming data in an IoT environment will result in a false positive events and poor quality of decisions

and actuation.

1.2 Motivation

In general, enhancing the IoT streaming data with semantics is a step to revolutionize the performance of IoT-based reactive services and data-driven decision-making processes for relevant web-based applications. This is because the recent paradigm shift in IoT technologies for the realisation of monitoring and controlling system aimed at achieving accurate real-time reactive services such as decision making and incident response activities within the domain of application can be guaranteed. Many of these real-time reactive services currently deployed in smart manufacturing process (Bi et al., 2014; Wang et al., 2004), process control in smart home automation(Wang et al., 2013) or environmental monitoring systems(D'Aniello et al., 2018; Dividino et al., 2018; Kamilaris et al., 2016) are allowed to make real-time decisions based on the IoT/sensor streaming data without proper quality validation.

These streaming data are known to be confronted with several quality problems relating to inconsistent (redundancy or noise), incompleteness (Missing data) and plausibility (Cross Sensitivity) (Ang et al., 2017; Barnaghi et al., 2015; Karkouch et al., 2016), thereby compromising the accuracy of the data points in IoT-based data-driven decision support systems and reactive applications subscribing to them. For example, the quality of output of such reactive services can also produce false-positive results (e.g. false fire alarm) or erroneous decisions in the presence of inconsistent or missing readings from sensors or related IoT nodes (Brown et al., 2019), hence making the system's efficiency to be compromised at run-time.

The rule-based taxonomy (Li et al., 2011) have identified the use of validity rules as one of the methodologies for ensuring high data quality in enterprise applications. Specifically, the data validity rules targets similar stream data quality relating to incompleteness, inconsistency and plausibility of data. Also, the IoT research community has continue

to provide technologies to ensure seamless integration of physical devices with reactive IoT applications without envisaging IoT streaming data quality issues. Providing a run-time validation and analysis of the IoT streaming data can guarantee effective and efficient controlling, monitoring, alerting and fault detection in a situation that requires rapid response to critical events within the smart IoT environment.

To deal with the IoT streaming data quality issues, requirements for a typical Semantic IoT Streaming data validation system are identified. These requirements are considered when developing the proposed semantic approach and its resultant framework for run-time validation of IoT streaming data, to detect quality issues relating to incomplete, inconsistent and plausible IoT/sensor streaming data. Consequently, the unified framework combines semantic stream processing and reasoning techniques capable of integrating the background knowledge with the streaming data. The reasoning system of the proposed framework is built with enhanced Jena Inference subsystem to guarantee continuous semantic reasoning process over IoT streaming data. The approach can produce inference while emphasizing on the expressivity of data representation and time requirement for IoT-based reactive applications and data-driven decision making in the IoT environment.

1.3 Research Aims and Objectives

Prompt by the motivations in the previous section, the main goal of the research is to provide an effective and efficient semantic validation of IoT streaming approach with a unified framework for IoT streaming data quality at run-time. Effective validation refers to the vision of the system to produce desired results in terms of stream relevance and accuracy metrics of the semantic validation approach evidenced by establishing that the predetermined specifications for the IoT streaming data are met. Similarly, Efficient validation focuses on the time performance of the validation framework by specifically considering the semantic reasoning time, processing time and latency of the semantic validation approach. Accordingly, the aim of this thesis is to convey the

features and data quality requirements of the IoT streaming data validation through a unified Semantic IoT Streaming data Validation framework. In this thesis, the semantic framework is a layered structure that combines semantic data modelling approaches, IoT streaming data ontologies, semantic reasoning, semantic rule selection algorithm and system requirements for semantic validation. Specifically, this aim is realised through the following objectives.

1.3.1 Objective I: To Establish the Relationship Among the IoT Streams Quality Problems

The fact that sensor streams are prone to quality issues is a major concern for researchers and smart solution providers when deploying an effective solution. Previous research findings have defined these sensor stream quality problems in a different context with different interpretations. It has resulted in continuous improvement of statistical approaches to combat the problems in near real-time.

The first object is to define a relationship for the quality issues such as: data ambiguity. inconsistency and incompleteness, relating to IoT/sensor streaming data. This relationship will provide a foundation for tackling the IoT stream quality problem as a holistic approach, and will provide a better understanding of IoT stream quality problem as consequent of other related quality issues. Hence, it will provide an unambiguous interpretation of the common sensor data quality issues.

1.3.2 Objective II: To Develop a Method for Effective IoT Resource and Sensor Streams Specification and Integration

Despite the wide availability of sensors and domain ontologies, inadequacies in the construction and vocabularies of the ontologies in the specific domain of applications still exist. Besides, they are only suitable for managing static knowledge, which renders them less relevant to frequent changes in domain applications with dynamic streams. Most of the available sensor ontology models are either lacking in the specification of

sensor streaming data or categorisation of specific domain related resources.

The second objective will provide a lightweight and evolving ontology model that can be translated into RDF graph by adopting the ontology design best practices. The ontology model will be beneficial for the persistence and management of previously validated IoT streams. It will also provide a means for annotating IoT streaming data at run-time. The ontology model will perform update upon addition of new IoT node or arrival of new data stream and maintenance of domain knowledge.

1.3.3 Objective III: To Develop a Method for the Continuous Semantic Reasoning of IoT Streaming Data

In the past years, several semantic reasoning engines have been developed and applied to process semantic web data including ontology models, for the purpose of producing new knowledge. However, these reasoning engines and corresponding subsystems have been applied only to static data held within the knowledge base but are currently lacking in their ability to provide support for run-time streaming data and continuous semantic reasoning with custom production rules.

The third objective will focus on adopting a semantic rule language that can support many of the RDF serialised data formats and can be coupled with RDF stream processing system to realise a forward chained inference. This approach will facilitate continuous reasoning of the semantic IoT streaming data produced from raw data streams. Consequently, it will also improve the performance of reactive IoT services and data-driven decision support systems in smart city applications.

1.3.4 Objective IV: To Integrate the Developed Semantic Methods with Mechanisms for Semantic validation of IoT streaming Data into a Unified Approach and Framework

Currently, it is rare to find a semantic stream processing system that combines the generation of logical/semantic streams from raw IoT streaming data, with semantic querying and reasoning function. Targeting the gap in fulfilment of semantic streaming data validation, the fourth objective is to integrate the aforementioned components and develop a unified framework and prototype implementation.

An API call is used to perform the continuous IoT streaming data validation tasks by IoT-bases reactive systems. A mapper from IoT middleware and connection infrastructure is required for invoking services for generating equivalent machine process-able data.

1.3.5 Objective V: To Conduct Case Study Evaluation with focus on RDF Serialised Data Formats

To demonstrate the feasibility and evaluation of the approach, the last objective of the thesis is to apply some real-life scenarios in domain of smart city and smart space with evidence of poor quality sensor/IoT data streams in various experimental runs. The experiments will provide a platform to critically examine the proposed approach with resultant framework and prototype implementations. Considering the possible range of alternative RDF data serialised formats for sensor streaming data representation/-modelling, the experimental results is expected to provide comparisons under various operating conditions of IoT nodes with associated stream quality problem.

1.4 Research Questions

Considering the research objectives in section 1.3, a number of research questions have been presented as follows:

- **RQ1:** How can the smart space domain and sensor network ontology support the incremental update of IoT streaming data caused by an addition of new data instance?
- **RQ2:** In what manner can the native Jena rule be adapted to support real-time an continuous time-aware reasoning?
- **RQ3:** How can the RDF stream processing system be combined with a forward chained rule to achieve the semantic reasoning and quality validation task of IoT streaming data in a run-time manner?
- **RQ4:** Can the Semantic validation approach with the reference framework be applied to detect inconsistency of IoT streaming data in smart home and air quality datasets?
- **RQ5:** To what extent is the effectiveness and efficiency of the semantic validation approach with focus on the serialised RDF data formats?

1.5 Contributions to Knowledge

The research presented in this thesis enhances the existing popular RDF stream processing system to support continuous reasoning and investigates its suitability in smart applications within the context of data-driven reactive services and decision-support systems. Further, it achieves this by providing a unified semantic-driven data validation approach with its framework for run-time validation and detection of quality issues in IoT streaming data. As result, a series of contributions have been identified. Specifically, these contributions and how they satisfy the research questions are highlighted as follows:

- Regarding **RQ1**, a lightweight IoT domain and sensor streaming data Ontology (SmartSUM) with embedding and evolving functionality was developed. In contrast to other Semantic Sensor Ontologies, SmartSUM is built from re-engineering

exiting related ontology models and enhanced such with an evolving feature to support the dynamic nature of the IoT streaming environment

- Generic stream quality validation rules and continuous Time-Aware Reasoning system based on enhanced RDF stream processing system for IoT streaming data. This provides answer to **RQ2** as the stream quality validation rule extends the Jena rule syntax with both time component and streaming window attribute of IoT streaming data. In attempt to provide answer to **RQ3**, the continuous reasoning engine have been layered with the popular Continuous Simple Protocol And RDF Query Language (Barbieri et al., 2010c) to perform run-time semantic inference.

- The Semantic IoT Streaming Data Validation Approach (*SISDaV*) for Plausibility, Incompleteness and Inconsistency in Streaming Data. By consulting previous works on related aspects of semantic modelling, Semantic and RDF stream processing systems including semantic processing techniques, an integrated framework is constructed while combining the above derived semantic techniques. *SISDaV* framework provide a unified semantic approach for pre-processing, modelling, reasoning and validating IoT streaming data quality requirement for reactive applications and data-driven decision making systems in web-based applications and related IoT environment. This provides answer to **RQ4** by facilitating the development of domain specific framework.

1.6 Structure of Thesis

The remainder of this thesis is organised as follows: Chapter Two provides the background and the related work relevant to the research in this thesis.The details of the general taxonomy of data uncertainty as well as the taxonomy for IoT streaming data are discussed. section 2.4 provides relevant semantic technologies related to the research with emphasis on data modelling, RDF stream processing systems (RSPs), semantic reasoning and ontology. The details of the applied evaluation metrics based on related work is introduced and explained in section 5.2.

Chapter three provides the design and implementation of SmartSUM model. It provides unambiguous concepts specification with data stream quality dimensions. It is used to annotate the IoT streaming data and enhance the semantic streaming data validation approach. The unique evolving feature of the model is discussed in section 3.4.

Details of the novel unified semantic approach and its resultant framework are discussed in chapter five. Section 4.3 provides details of the semantic approach while the resulting framework is described in section 4.4.

An evaluation of the novel unified semantic approach and its resultant framework is presented in Chapter six. The evaluation case studies, including a description of the IoT streaming/sensor data sets, used to test effectiveness and efficiency of the approach are described in Section 5.3 and 5.7 respectively. Section 5.8 later details general discussions of results from experiments involving the case studies.

Finally, Chapter seven summarises the thesis by presenting the conclusions and future directions of the research.

1.7 List of Publications

Parts of the research have been published.

- Bamgboye, O., Liu, X., & Cruickshank, P. (2018). Towards modelling and reasoning about uncertain data of sensor measurements for decision support in smart spaces. *In 2018 IEEE 42nd annual computer software and applications conference (COMPSAC)* Vol. 2, pp. 744-749.

- Bamgboye, O., Liu, X., & Cruickshank, P. (2019). Semantic Stream Management Framework for Data Consistency in Smart Spaces. *In 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)* Vol. 2, pp. 85-90

Second Chapter

Research Background and Related Work

2.1 Introduction

The Semantic IoT Streaming data Validation approach and its Framework are based on the fundamental aspects of Semantic Stream processing and semantic web technologies. In this chapter, the general taxonomy of data uncertainty problem as it relates to data quality problem is presented in 2.2. In attempt to satisfy the first objective of the study, a taxonomy of uncertainty in IoT streaming data with data quality requirements. Approaches to various sensor data quality issues are discussed in 2.3. Section 2.4 provides related research in semantic technologies with emphasis on data modelling, RDF stream processing systems (RSPs), semantic reasoning and ontology. Semantic stream Validation requirements and how these requirements are satisfied by RSPs are presented in 2.5 and 2.6 respectively. Section 2.7 provides background insights into smart city models in relation to its application to smart spaces, while section 2.8 concludes the chapter.

2.2 Taxonomy of Data Uncertainty

Generally, data uncertainty describes a situation where human or machine (software agent/system), only possess partial knowledge about the truth-value of a given piece of data. The categorisation of data uncertainty can also perceived as either objective or subjective (Kläs & Vollmer, 2018). The objective uncertainty in data refers to the degree

of the probability about the truth of a particular data whereas the subjective refers to the agent's opinion about the truth-value. Subjective uncertainty is also considered as most common data and it describes the degree of deviation from accuracy, completeness and consistency of measurement data. A number of classifications have been established in various domain in order to provide better understanding of the data quality problems

Figure 2.1 describe the current classification of uncertainty in a related sensor-based

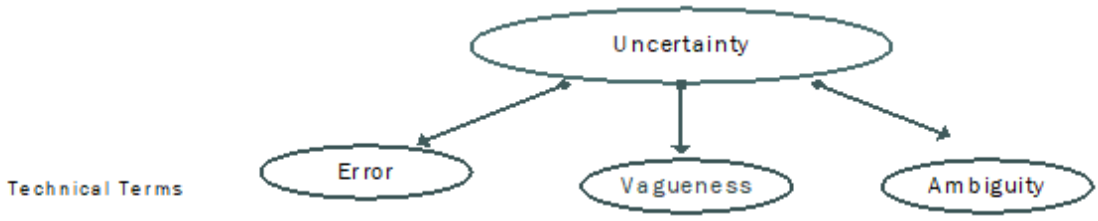

Figure 2.1: Classification of Data Uncertainty in GIS

information systems (Jekjantuk et al., 2016). The classification attributed the sensor anomalies to data uncertainty, which can often be expressed as error, vagueness and ambiguity in measurements within the Geographical Information Systems (GIS). The notion of ambiguity and vagueness have been considered to be equivalent in the initial classification for intelligent environments (Gu et al., 2004). The quality problem relating to vagueness can be viewed as either qualitative or quantitative (Kumar et al., 2006). Vagueness is quantitative in sensor observation when there is lack of precise boundaries along one or more quality dimensions (e.g. High Temperature, cold weather). It is considered qualitative if there exist a variety of conditions, which makes it impossible to make any crisp identification of those combinations that are sufficient for an application e.g. the term Season can contain a qualitative vagueness, this is because there are several weather conditions that can determine a particular season in the year. In the same perspective, source of uncertainties in sensor measurement are related to inconsistencies, ambiguity, imprecision and noise (Leyk et al., 2005). Furthermore, the taxonomy for uncertainty in sensor data at database level is seen in (Almeida & López-de-Ipiña, 2011). It comprises of the point uncertainty, interval uncertainty and probabilistic uncertainties.

Most of the existing classifications are yet to provide the holistic view of the uncertainty problem as it relates to IoT and smart spaces. In the present research involving IoT including smart spaces, subjective uncertainty is the most common data quality problem. This is also perceived as errors in distributed sensor readings or measurements within smart spaces. Providing an explicit description of the uncertainty problem in relation to IoT streaming and smart spaces will facilitate the development of a more holistic approach to solve the quality problems.

2.2.1 Taxonomy of Data Uncertainty in IoT Streaming Data

The desire to improve service construction of smart spaces continues to grow with the aim of achieving an efficient ubiquitous computing environment, even as the cost of sensors continues to remain relatively cheap. The high volume of data produced in this environment is mainly generated from a range of IoT nodes including sensors. The sensors itself can represent one of the main sources or contributors to data quality issues (Karkouch et al., 2016).

Generally, data quality issues in IoT/sensor streams occurs at the instance-level, which can collectively be described as data uncertainties. The definition emanates from the ambiguous interpretation of the data at lower level of abstraction during run-time before it is eventually interpreted by a sensor instrumentation or the data analytic engine. Uncertainties in IoT and sensor streams are considered to result from several quality problems that include imprecision, noise, ambiguity and inconsistency of readings or measurement values (Kumar et al., 2006). In both Smart Space and IoT domain, this research describes uncertainty as the degree of deviation from accuracy, completeness and consistency of IoT streaming data. It perceives the deviation from accuracy to be an indication of the presence of vagueness or plausibility. This is because a vague or plausible IoT/sensor data can often be true in it own sense but usually lacking clear boundaries during interpretation by a subscribing systems. The category of uncertainty

proposed in this section can be considered to be window-based. This is because it is possible to experience each type of quality issue within separate windows in a particular streaming cycle.

Similarly, the taxonomy of IoT data quality problems have been used to describe uncertainty within the smart spaces (Bamgboye et al., 2018). Figure 2.2 considered the uncertain data from sensors and IoT nodes as transient and also provided an holistic view of the related data quality problems in this domain. It provide a way to take the peculiarity of IoT/sensor data into account by considering the temporal characteristics of the data and how it can quickly become plausible based on expired streaming window session. The hierarchical relationship of the quality issues in the taxonomy has been established based on various classifications from relevant literatures(Anja, 2009; Cheng, 2003; Karkouch et al., 2016; Kumar et al., 2006; Sta, 2016) in sensor data.

The taxonomy clearly shows the category of uncertainty in IoT and sensor streams

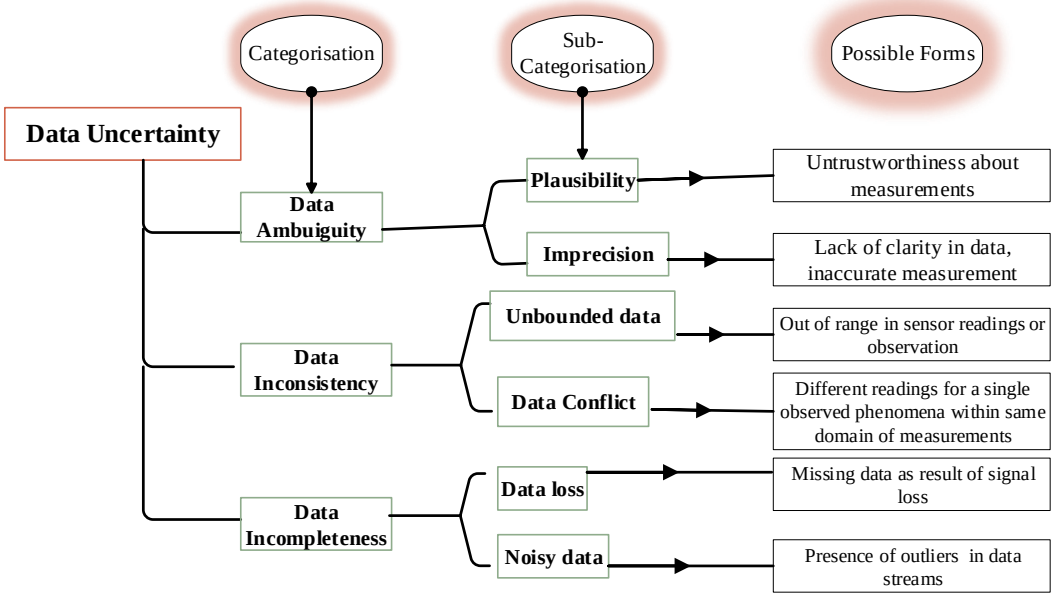

Figure 2.2: Taxonomy of Uncertainty in IoT Streaming Data

along the dimension of Data Ambiguity, Inconsistency and Incompleteness of readings or measurements. The description of the possible causes of the uncertainty problems

is discussed in table 2.1. Inconsistency and Incompleteness in IoT streaming data are considered to occur at the lower observing layer of the sensor readings while Data Ambiguity is a quality problem often experienced at the higher contextual level of smart space applications. Earlier study (Elnahrawy & Nath, 2003) associated quality problems relating to imprecision, inconsistency and noise to random errors. This type of errors exists distinctively in smart spaces. For example, inconsistency can arise when two sensors of the same type deployed within the same space give different readings. We define this case of the problem as data conflict in our proposed taxonomy. In a similar perspective, the investigation by (Tan et al., 2005) suggests incompleteness in sensor reading is mostly caused by sensor failure which often results in data loss. Commonly, the majority of the IoT and sensor connected devices will represent missing or incomplete data streams with specific data string format output. IoT streaming data produced from multiple sources are available with several inconsistencies due to the nature of structured, semi-structured and unstructured formats (Mishra et al., 2015; Rao, 2019; Vongsingthong & Smanchat, 2015).

The accuracy and reliability of sensor measurement are dependent on the technique for managing the plausibility and related quality issues in sensor readings to improve Smart City and smart space experience (Kuemper et al., 2016). However, defining a kind of holistic approach to solving these issues may require setting out some data quality requirements for intelligent or Smart Spaces where the IoT nodes including sensors will play an active role in data provisioning.

2.2.2 Data quality Requirements for IoT Nodes and Smart Spaces

Data remains an essential ingredient for categories of automated systems such as decision support systems and other critical systems that heavily depends on sensor readings in delivering related services. It is therefore important to ensure the data quality requirements are satisfied to deliver high quality services. These automated systems must then agree data meets one or more of the following quality requirements,

Table 2.1: Description of uncertainty problems in IoT Streaming Data

Quality Problem	Description	Appearances/Possible causes
Data Ambiguity	Data ambiguity describes data quality problem in terms of false interpretations arising from inaccuracy due to inexactness contexts or measurement values (Eppler, 2006; Paulus et al., 2019).	Observation measurement having blurred boundaries disallowing precise or clear distinction between values or data
Plausibility	Forged or nearness in the agreement between the expected value of measurement and the accepted reference value because of imprecise or uncertain IoT node (Bißmeyer et al., 2012).	Mostly caused by false positives or impreciseness in values
Imprecision	The degree of inexactness or inaccuracy of agreement arising from uncertainty between measurement quantity values (Cheng et al., 2003; Dhillon et al., 2002)	Obtained as a result of replicate measurements on the same or similar object under specified conditions
Incompleteness	It describes the extent to which data is missing and insufficient or inconsistent in breadth and depth for the task and purpose at hand, thereby resulting in uncertain information (Pipino et al., 2002; Tan et al., 2005).	Totally or partially Missing value/data caused by network failure or data damage
Inconsistency	It arises whenever there are incorrect or different versions (redundancy) of the same data or measurement value due to uncertainties based on ambiguous and noisy data streams (Kumar et al., 2006)	It can be due to data generated at different time snapshots, or different sources or even abstraction level.

which are derived from a review of relevant literature. In particular. the data quality problems in table 2.1 remains the major impediments in the fulfilment of these quality requirements, which are also considered to be essential as part of the smart space

components.

- **Accessibility** refers to the availability and ease of IoT streaming data exploitation/retrieval within the space despite the heterogeneous or multi-modal characteristics of devices and data. The current deployment of sensors in smart space allows each separate or individual sensor to generate its data format unilaterally without a common interface for accessing similar or other heterogeneous data. It will be necessary to provide an infrastructure for common exploitation/retrieval of heterogeneous streaming data with the same protocol.

- **Accuracy** is the degree at to which sensor readings represents the measured phenomena and the extent to which value v from sensor readings belongs to a closed interval of $-\beta \leq \upsilon \leq \beta$, for the absolute systematic error β. It tells how closely the output of an IoT node or sensor reading from an instrument or device corresponds to its 'true' value. For example, a sensor calibrated at < ± 0.1% of measurement will mean the actual reading will be applied to ± 0.010 units of measurement or less. This means that any variation between the 'true' values is referred to as 'error'.

- **Completeness** is the extent to which an IoT node producing sensor readings at specific data points contains no missing data point including the timestamp and, are sufficient in the ratio of the breadth and depth of the data within a particular streaming window. For example, an active sensor may suffer from intermittent signal or network failures thereby registering wrong values for these data points. It is therefore imperative to ensure the missing data points are not delivered as part of complete realistic measurements.

- **Consistency** refers to the extent to which data produced at IoT nodes by a particular or group of sensors are available in the same format without undue repetitions or redundancies within same timestamp and window for subsequent collaborative or intelligent processing. As an illustration, multiple numbers of sensors of the same type and measuring the same phenomena can be placed in the same

domain while providing the individual readings to a specific application. It is important for each instance of measurement to be uniquely identified before its consumption by an application.

- **Interpretability** refers to the degree of the appropriateness and clarity of data in terms of meaning and format. It is noticed not all sensor interprets reading in understandable forms easily for machine-to-machine communication e.g. some sensor to interpret measurements as "0" and "1" while others can interpret using integer and real number values. Also, it can result in efficient processing and makes more sense if the meaning of this measurement is part of streaming data processing.

- **Implausibility** plausibility refers to the extent of which a given IoT node or sensor reading is considered as true and acceptable or credible for a given measurement. For instance, it may be important to ascertain that a sensor such as a temperature sensor is actually giving the reading of the space it is being deployed and not the temperature of the human body or other objects within the Smart Space. Implausibility emphasises on the quality of IoT data streams to possess unambiguous representation of the data.

- **Timeliness** represents the requirement for IoT node to be able to process streaming data in a timely manner and differentiate the actual timestamp of streaming data from the registering timestamp of IoT processing node during measurement. Also, the interval of time between when the sensor measures the real-life event and is available for use by other smart space components should be minimal.

There has been continued efforts to improve the performance and efficiency of smart computing systems that rely on sensor streaming data. Many of the efforts separately concentrate on improving the connectivity of IoTs and quality of sensor data. The approaches for sensor data processing are categorised into two different types that are based on statistical and semantic techniques. In the subsequent sections, we described these techniques in relation to how they have been applied to sensor streams.

The next sections focus on the various techniques and approaches to deal with the quality problems in line with satisfying the quality requirements of IoT streaming data.

2.3 Approaches to Sensor Data quality Problems

There have been several attempts to improve the quality of data produced at IoT node including sensor device in an attempt to reduce the level of uncertainty associated with streaming data. Though the semantic approach to processing IoT/sensor data is still at the early stage of research, there have been a number of statistical techniques applied to solve the quality issues in sensor data. In the general perspective, these attempts or approaches to data quality in sensor measurements are described as statistical approaches.

The problem relating to missing data points in sensor measurement has often been addressed through the method of statistical interpolation (Appice et al., 2013; Yoon et al., 2018) while other data quality-related problems have been addressed with traditional statistical approaches such as regression analysis. All these approaches are shown to have fallen short in the ability to capture the known challenges of velocity and variability that defines the nature of IoT streaming data. Besides, these approaches are unable to provide data integration, which is required to leverage interpolation by inferring missing values based on other or available IoT or sensor readings (Karkouch et al., 2016).

The adoption of the Bayesian approach and Gaussian distribution was used to compute an estimate of accuracy in sensor readings at base station (Elnahrawy & Nath, 2003). In particular, errors in form of the inaccuracy of measurements, imprecision of observations, and environmental factors are attributed to noise. The approach includes an uncertainty model for cleaning and interrogating sensor readings. The approach, however, ignores the aspects of inconsistency and incompleteness that may be predominant at the sensor node. A similar problem of noisy and incomplete sensor streams was also managed with the use of the Kalman Filter and regression models (Tan et al., 2005) to perform cleaning of noise and eventual interpolation of incomplete on raw

sensor data. Though the approach was introduced to improve the limitation of the traditional data cleaning method, it is still inadequate for managing inconsistency in the sensor data. To address this problem, a multisensory data fusion that combines modified Bayesian fusion with Kalman filtering algorithms was proposed (Abdulhafiz & Khamis, 2013). In the context of smart city applications, sensor data quality problems related to inaccuracy and, being a subset of data uncertainty have been subjected to a statistical model involving the use of Dempster-Shafer Theory (DST) combined with the evidential databases (Sta, 2016). Though this approach suggests being promising in dealing with noise, it does not prove to be applicable for run-time or continuous reasoning of sensor streams and fails in achieving data interoperability.

The use of metadata to improve the accuracy and correctness of retrieved sensor values with spatial-temporal characteristics at database level has been implemented with the introduction of enhanced Probabilistic threshold query (Prabhakar, n.d.). The approach is strictly applied to sole the related data quality problems in database system and does not clearly represents the meta-data approach within the semantic web. One step in that direction is to consider the use of a semantic approach in managing the quality issues of sensor data to support other related applications (Calder et al., 2010).

The Semantic data processing approaches are becoming popular in the IoT domain. Various approaches consider the use of knowledge (or knowledge graph) representation in combination with linked data and reasoning techniques. The knowledge representation techniques involve the application of standard semantic model languages such as the OWL, RDF and Frame-based languages. RDF has become the most popular and widely used language for data representation and data exchange across semantic-based and web-based systems. The use of the RDF data model in the processing of sensor streaming data provides different levels of expressivity, especially with different serialisation formats.

2.4 Semantic Web Technologies

Semantic Web - the initiative of Tim Bernes-Lee, is built on standards to enable the extension of the Web with machine-interpretable meaning, hence facilitating data integration/sharing and interoperability amongst interconnected machines (Berners-Lee et al., 2001). The concept of semantic Web is based on the formalism of the Resource Description Framework (RDF). This allows the linking and merging of relations between entities from multiple resources on the Web through the use of IRI. The common vocabulary recognised by the RDF to model and provide a description for data are the RDF Schema (RDFS) and ontology. Semantic technologies can enable smart objects to interact in an intelligent manner with each other on IoT. The application of these technologies to IoT systems automate data/information acquisition and decision-making process, while enhancing the development of advanced applications.

2.4.1 Semantic Data Representation with RDF

Semantic data representation is often employed in the description of semantic web resources which may also include certain data manipulations. The description of the RDF data or resources is available as *Property-Value* often referred to as *Triples* or RDF statements, which provides logical representations of data as a network of graph (also called Knowledge Graph) in semantic web applications(Beckett & McBride, 2004). RDF statement contains elements that is of the form *S, P, O* where each element corresponds to *Subject*, *Property* and *Object* respectively. Each of the element is processed using the RDF signature similar to $(X \cup Y) \times X \times (X \cup Y \cup Z)$ where X, Y and Z may represent a set of respective URI resources that points to the nodes and RDF literals.

RDF provides rule-based axiomatization of the RDFS semantics that can be executed over any RDF graph. RDF data can be represented using the alternative W3C standard for publishing and exchanging semantic data. These formats include RDF/XML, Turtle, N3 and N-Triples, representations. These formats often differ in terms of expressivity

and resource usage (Su et al., 2015) as it can be seen in various domains of application. Similarly, both issues are envisaged to place a major constraint on the semantic IoT streaming data validation approach, where time plays a major role. Therefore, it is worthwhile to compare the RDF data formats to understand which will support the semantic validation approach effectively and efficiently.

The serialised RDF data models mostly adopted by IoT systems is broadly categorised as XML-based and Non-XML based. The classification of the formats is based on their expressivity, simplicity and applicability. These formats are suitable for IoT streaming data modelling and related web applications in IoT domain (Maarala et al., 2017).

- **RDF/XML:** As previously discussed, RDF/XML serialised format enhances IoT

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:smartSpace="http://localhost:8080/smartSpace#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" >
 <rdf:Description rdf:about="http://localhost:8080/smartSpace#pressureReading8">
         <smartSpace:hasPressureReading rdf:datatype="http://www.w3.org/2001/
                        XMLSchema#float">752.17</smartSpace:hasPressureReading>
        <smartSpace:pressureHasTimestamp rdf:datatype="http://www.w3.org/2001/
                        XMLSchema#dateTime">2020-01-20T09:42:12.084Z</
                        smartSpace:pressureHasTimestamp>
        <rdf:type rdf:resource="http://localhost:8080/smartSpace#pressureValue"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://localhost:8080/smartSpace#pressureReading10">
         <smartSpace:hasPressureReading rdf:datatype="http://www.w3.org/2001/
                XMLSchema#float">753.76</smartSpace:hasPressureReading>
         <smartSpace:pressureHasTimestamp rdf:datatype="http://www.w3.org/2001/
        XMLSchema#dateTime">2020-01-20T09:42:22.085Z</smartSpace:pressureHasTimestamp>
         <rdf:type rdf:resource="http://localhost:8080/smartSpace#pressureValue"/>
 </rdf:Description>
```

Figure 2.3: RDF/XML Listing

processing node to parse, store and serialise XML data in a simplified manner. The listing in figure 2.3 shows a snapshot of RDF/XML serialisation format for IoT streaming data. In this case, a sensor streaming data with sliding windows

is represented as quadruple statements that contain the equivalent <*Subject, Predicate, Object, Timestamp*> of individual sensor streaming data within the RDF graph. The listing indicates the combination of two different structures: a tree structure and quadruple-based graph. The possible implication of the underlying structures is that it can make serialisation more complex and too wordy when compared with the other serialisation standards (Su et al., 2015). Besides, the regular RDF/XML format does not clearly show the triple pattern in static data and hence it may only be useful when the developer is only interested in working with XML.

- **Notation3(.n3) :** The n3 serialisation was introduced to provide better expressiveness as against the RDF/XML format. This can guarantee a more compact and readable alternative for serialising real-time sensor streaming data. Considering the listing in figure 2.4, n3 serialisation improves the readability of sensor streaming data and allows easy integration with semantic rules for reasoning purposes during the sensor streaming data validation process. These prospects are realised through the use of URI abbreviations that are bounded to a namespace, appropriate quantification that allows rules to be expressed, and simple grammar with a measure of consistency. The n3 is capable of encoding the streaming data statements as well as its meaning. Also, .n3 are supported as synonyms for the Turtle serialisation when implemented in java programs.

- **Turtle (.ttl) :** The terseness of Turtle serialisation format contributed to its wide popularity, which also made it to be considered as another alternative of RDF data. Being a subset of .n3 (figure 2.5) because of its support for *@prefixes*, the present research considers Turtle format as one of the potentials RDF serialisations for the IoT/sensor streaming data over a streaming window. The choice for Turtle is inspired by its ease to generate semantic data streams, and support by popular tools and common APIs including Jena and OWL API (Horridge, n.d.). Though the serialisation of static data using Turtle as modelling choice shares limitation with the RDF/XML format in that it is more expensive to parse unlike the N-Triple

```
@prefix owl:   <http://www.w3.org/2002/07/owl#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix smartSpace: <http://localhost:8080/smartSpace#> .

smartSpace:humidityMeanValue
     a      rdfs:Class .

smartSpace:tempReadings9
     a                    smartSpace:tempValue ;
     smartSpace:hasId          "Sensor 1"^^xsd:string ;
     smartSpace:hasSeason       "winter"^^xsd:string ;
     smartSpace:hasValue       "23.53"^^xsd:float ;
     smartSpace:tempHasTimestamp  "2020-01-
23T18:06:16.263Z"^^xsd:dateTime .

smartSpace:humitidyReadings11
     a                     smartSpace:humidityValue ;
     smartSpace:hasHumidityReading  90 ;
     smartSpace:humidityHasTimestamp
          "2020-01-23T18:06:26.249Z"^^xsd:dateTime .

smartSpace:tempValue  a  rdfs:Class .

smartSpace:tempReadings12
     a                    smartSpace:tempValue ;
     smartSpace:errorData       "Missing Value"^^xsd:string ;
     smartSpace:hasId          "Sensor 1"^^xsd:string ;
     smartSpace:hasSeason       "winter"^^xsd:string ;
     smartSpace:hasValue       "8888.88"^^xsd:float ;
     smartSpace:tempHasTimestamp  "2020-01-
23T18:06:31.265Z"^^xsd:dateTime .
```

Figure 2.4: N3 Listing

and N-Quads counterpart, It will be interesting to measure the impact of this limitation on reasoning with dynamic data such as the sensor streaming data in a fast dynamic environment.

- **N-Triple (.nt):** The suitability of N-Triple for machine-to-machine communication is one of the reason for consideration in serialising the sensor streaming data at run-time. It is considered suitable for storage of millions of triples especially, in

```
@prefix owl:   <http://www.w3.org/2002/07/owl#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix smartSpace: <http://localhost:8080/smartSpace#> .

smartSpace:temp2Readings5
      a                    smartSpace:tempValue ;
      smartSpace:hasId          "Sensor 2"^^xsd:string ;
      smartSpace:hasSeason       "winter"^^xsd:string ;
      smartSpace:hasValue        "18.35"^^xsd:float ;
      smartSpace:isInconsistent    "Erroneous reading" ;
      smartSpace:tempHasTimestamp  "2020-01-20T09:51:55.005Z"^^xsd:dateTime .

smartSpace:tempReadings6
      a                    smartSpace:tempValue ;
      smartSpace:errorData       "Missing Value"^^xsd:string ;
      smartSpace:hasId          "Sensor 1"^^xsd:string ;
      smartSpace:hasSeason       "winter"^^xsd:string ;
      smartSpace:hasValue        "8888.88"^^xsd:float ;
      smartSpace:isInconsistent    "Erroneous reading" ;
      smartSpace:tempHasTimestamp  "2020-01-20T09:52:00.040Z"^^xsd:dateTime .

smartSpace:temp2Readings3
      a                    smartSpace:tempValue ;
      smartSpace:hasId          "Sensor 2"^^xsd:string ;
      smartSpace:hasSeason       "winter"^^xsd:string ;
      smartSpace:hasValue        "27.18"^^xsd:float ;
      smartSpace:isInconsistent    "Erroneous reading" ;
      smartSpace:tempHasTimestamp  "2020-01-20T09:51:45.004Z"^^xsd:dateTime .
```

Figure 2.5: Turtle Listing

the case of IoT streaming data where a large number of data will require to be processed. Serialising each element of the sensor streaming data with an individual timestamp as indicated in figure 2.6 is also one way of verifying its suitability for IoT web-based applications. N-Triple is known for its high performance in parsing common static data. The N-triple is known to have the potential to used by webs services and clients systems that consume the data.

2.4.2 RDF Stream Processing Systems

The research in the field of semantic stream processing particularly, RDF Stream Processing (RSP) systems have continued to evolve in two aspects: providing support for semantic representation of temporal attribute of the data streams and developing an approach to allow continuous semantic query during processing the data in motion. Majority of the semantic stream processing system currently developed are based on the native SPARQL query language and RDF data model. In principle, these systems extend

```
<http://localhost:8080/smartSpace#humitidyReadings11> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://localhost:8080/smartSpace#humidityValue>.
        <http://localhost:8080/smartSpace#humitidyReadings11> <http://localhost:8080/
        smartSpace#hasHumidityReading> "69"^^<http://www.w3.org/2001/
XMLSchema#integer>.
        <http://localhost:8080/smartSpace#humitidyReadings11> <http://localhost:8080/
        smartSpace#humidityHasTimestamp> "2020-01-20T10:14:10.811Z"^^<http://
www.w3.org/2001/XMLSchema#dateTime>.
<http://localhost:8080/smartSpace#tempReadings9> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://localhost:8080/smartSpace#tempValue>.
        <http://localhost:8080/smartSpace#tempReadings9> <http://localhost:8080/
smartSpace#hasValue>   "17.48"^^<http://www.w3.org/2001/XMLSchema#float>.
        <http://localhost:8080/smartSpace#tempReadings9> <http://localhost:8080/
        smartSpace#tempHasTimestamp> "2020-01-20T10:14:00.824Z"^^<http://www.w3.org/
2001/XMLSchema#dateTime>.
        <http://localhost:8080/smartSpace#tempReadings9> <http://localhost:8080/
smartSpace#hasId> "Sensor      1"^^<http://www.w3.org/2001/XMLSchema#string>.
        <http://localhost:8080/smartSpace#tempReadings9> <http://localhost:8080/
smartSpace#hasSeason> "winter"^^<http://www.w3.org/2001/XMLSchema#string>.
```

Figure 2.6: N-Triple Listing

SPARQL query with a streaming operator to achieve continuous query and RDF data model with time annotations for proper semantic representation of data stream. The differences in the semantics and syntax of various stream query languages necessitated the introduction of a unified query model proposed by (Dell'Aglio et al., 2014). The subsequent paragraphs consider the details of the semantic stream processing systems from the perspectives of the RDF and SPARQL extensions.

Most works in the field of semantic stream representation perceive data streams as an unbounded sequence of data with individually annotated timestamps. This description follows same from the field of DSMS. Apart from INSTANS (Rinne et al., 2012b) that implicitly handles the time component of data streams within the schema, other RSPs including Streaming SPARQL (Bolles et al., 2008), C-SPARQL (Barbieri et al., 2009), CQELS (Le-Phuoc et al., 2011), C-ASP (Pham et al., 2019), RSP-QL (Dell'Aglio et al., 2014) and $SPARQL_{Stream}$ (Calbimonte et al., 2010) uses the RDF statement to represent timestamp as single point associated to each RDF statement. An exception in the explicit time representation is EP-SPARQL (Anicic et al., 2011) and Streaming SPARQL which considers interval between timestamps to represent the time dimension and check the

validity of each corresponding RDF statements.

In terms of achieving continuous query, many of the proposed RSP with SPARQL extension adapt the relational Continuous Query Language Model (CQL). The adaptation of the CQL model involves the use of three categories of operators to process the RDF streams. The first operator is the window operator that is used to transform the RDF streams into a form that support mappings. This operator has been implemented a way that supports time-based and triple-based sliding in windows in existing systems such as C-SPARQL , CQELS , C-ASP, RSP-QL and $SPARQL_{Stream}$. Another operator adapted from CQL is the relation-to-relation operator that is used for the representation of SPARQL algebra. It allows the language extension to directly process the inputs and outputs without necessarily redefining the operators. The third category is relation-to-stream used in the mapping set transformation of previously produced class into RDF stream.

The type of execution strategy currently in use by the existing RSP varies between systems. Periodic update of input window is common to RSP systems such as C-SPARQL, Streaming SPARQL and $SPARQL_{Stream}$ while other RSP systems like CQELS, INSTANS and EP-SPARQL are mainly based on data-driven strategy. Systems based on window update will always re-execute the existing query against the subsequent new data streams window.

In general comparison, despite the effective implementation of the temporal operators, Streaming SPARQL still lacks aggregation function and the requirement for multiple stream query is currently not supported by the system. Both INSTANS and C-SPARQL are able to address this problem. The C-SPARQL library is built to support concurrent query over multiple streaming data and supports most of the alternative formats for RDF data serialisation format. However, the ability to perform reasoning over the continuous data point is still lacking in this system despite its ability to fully support temporal processing. Similarly, CQELS "white-box" approach for temporal processing

is able to achieve some degree of scalability and query optimisation but the approach is completed only on a fixed number of data streams. At higher abstraction level is the category of event processing systems such as EP-SPARQL that is also based on SPARQL extension. The time element used for the RDF processing is computed as the interval of time between events.This makes is rather impossible to understand the specific time-instant for each stream element.

Therefore, most of the RSP is still considered to be lacking in terms of providing reasoning functionality over RDF/OWL and streaming data at any level of application or context. Furthermore, the application of the RSP has been seen to be implemented at different systems' granularity levels (Bamgboye et al., 2019; Jajaga et al., 2015). Some systems apply the RSP at schema level while others implement them at event or data levels depending on the focus of the application. In this case, the application context can either be event-driven or data-driven applications. However, in the IoT domain including the Smart Spaces, where many of the applications rely on sensor network including streaming data, the RSP will be required at the data layer. In this research, the popular C-SPARQL will be enhanced to support reasoning based on it support for serialised RDF data formats and compatibility for rule-based reasoning.

2.4.3 Semantic Stream Reasoning

The Semantic Stream Reasoning is an emerging field that combines the approach used for continuous querying over IoT data streams with reasoning usually with SPARQL extension languages. The expressivity of the reasoning system refers to how to rule languages can represent various axioms and how those axioms are interpreted or processed by the reasoner. The reasoning methods imply the various strategies or approaches adopted by the reasoning system. The feature relating to soundness means the ability of the reasoner to produce all possible inference or speed-up reasoning.

The reasoning process is realised through a reasoning engine or reasoners, which are

described as software or tool that relies on rules to achieve reasoning tasks. Reasoners have had a long history of application support for deriving new facts from ontological models including knowledge graphs. The limitations of the current ontology-based reasoners concerning the computational complexity, inference with raw data streams and data validation of streaming data have been identified by (Aufaure et al., 2016; Lehmann & Völker, 2014). They are able to resolve redundancy and inconsistency among ontological concepts of TBoX (describing the specific domain class and properties) and static individual held within the ABox (describes the domain assertion). ABox reasoning is a feature of reasoner that allows inferences to be drawn from a set of an individual in a knowledge graph. Varieties of reasoners that currently exist can perform reasoning at schema level while using the RDFS and OWL vocabularies including most of the serialised RDF data formats.

Stream reasoning systems are considered effective only if they can consider the time elements and the window intervals of the continuous infinite streams by adopting a set of rules (Stuckenschmidt et al., 2010). The primary goal of performing reasoning is usually to generate new set of knowledge from existing data or information made available by the domain of interest during semantic inference. The new knowledge or information is later applied to solve a specific problem or used to trigger an event such as making a decision or reactive services. Unlike the statistical inference such as machine learning that trains a model on historical data before applying the model to process data, Semantic inference leverages the knowledge about data through data interoperability before it can be used for processing. In this way, the semantic inference technique will provide faster and efficient processing in a dynamic environment that data or information often changes over time. Though statistical inference involving machine learning is increasingly becoming popular, it requires more time to update the model with the new changes to data produced within the dynamic environment before it can be applied to solve the domain problems. Considering the differences between these reasoning/inference types, the choice of reasoning strategy for run-time IoT Streaming data and application should be determined by the ability to infer new

knowledge without necessarily having to re-train the model in a time-critical reactive domain. A semantic approach to sensor streams reasoning (Compton et al., 2009) and validation (Calder et al., 2010) of anomalies in sensor data has earlier been suggested for new knowledge discovery in sensor-driven applications. Both approaches do not reflect the capabilities to support streaming data, as it does not consider the temporal characteristics of the data.

This section explores the category of semantic rule-based reasoning developed for semantic stream processing by adapting relevant features earlier defined for ontology reasoning (Abburu, 2012), which include: expressivity, methods, soundness, support for ABox reasoning, incremental classification and, Jena support. Jena support describes the reasoning engine that is compatible with the Jena API or/and Jena rule language. The characteristics relating to incremental classification refers to the category of reasoner that allows Knowledge graph or ontology to be updated by classification that involves addition or removal of the newly produced assertion that results in the new hierarchy.
In the remaining part of the section, semantic reasoning have been considered in two separate categories that consist of the schema-level stream reasoners and Enhanced C-SPARQL Stream reasoning.

2.4.3.1 Schema-Level Stream Reasoning

The majority of the early stream reasoning system is based on the RDFS and OWL2 ontology language. For example, TelegraphCQ (Chandrasekaran et al., 2003) was developed to achieve RDF entailment regime with sets of continuous entailment rules to determine the subclass database events of the particular target event. The reasoning engine uses a black-box approach and presents low expressivity when dealing with the major RDFS properties (including *subClassOf, subPropertyOf, range, and domain*) and the OWL relationship (that is, *inverseOf*).

In an attempt to enhance the popular C-SPARQL RSP system with incremental reas-

oning capability the authors (Barbieri et al., 2010b) developed rule-based reasoning with forward chaining inference implemented with approach later described as IMaRS by (Dell'Aglio & Della Valle, 2014) to perform the incremental maintenance of materialisation for ontological entailment with RDF stream. The inference system uses the Jena Generic Rule Language and processes the streaming data based on the order of arrival. A set of the built-in function is used to add the expiration time to RDF statement to compute the materialisation. The system requires a fixed time window with DRed algorithm (Gruber et al., 1993) to manage the deletions of RDF triples that no longer exist within the current window to achieve complete materialisation.

An approach to scalable reasoning over ontology streams through a method of syntactic approximation was introduced in (Ren et al., 2010). The approach was introduced to facilitate stream reasoning in particular for expressive ontology by a combination of DRed algorithm with a syntactic approximation to achieve some degree of soundness, expressivity and tractability through entailment of logical axioms and consequences. The approach mainly considers providing incremental classification through the reasoning of concepts in the TBox and does not indicate support for external API such as the Jena API.

Sparkwave (Komazec & Cerri, 2012) is another incremental approach to reasoning based on the basic RETE algorithm (which adopts forward-chaining reasoning) and RDF/RDFS rules, developed for pattern detection of the schema describing RDF data streams. It provides leverage for RDF schema to compute the entailment but not for incremental classification of a new axiom or RDF stream. It performs the reasoning operation on an individual element of ABox while using fixed RDF schema with a preprocessing network called Epsilon that acts as the entry point into the RETE network. The entailment rules for the RDF/RDFS does not have an impact on RDF stream at run-time since it operates on fixed RDF schema. Though the primary aim of Sparkwave is to complement existing RSP, expressivity was not part of the focus and support for Jena API was not clearly stated. Besides, Sparkwave is considered to share similarity

with IMaRS in terms of limitation to support temporal and arithmetic operators which, also forms key aspects for run-time reasoning for streaming data.

Majority of the Schema-Level reasoning is adequate for performing reasoning on implement class/subclass subsumption on the element of TBox and inferred axiom or data element of ABox but are still inadequate in performing run-time reasoning on streaming data. Also, most reasoners developed with OWL-DL will require more computing resources and computational time (Thomas et al., 2010). One of the recent approaches of providing reasoning for streaming data currently involves the application of semantic rules at a higher level of context or applications.

The existing approaches comprising of lightweight and complex reasoning over ontologies and its schema is not adequate when it comes to performing reasoning task over raw IoT streaming data that has dynamic characteristics to change over time. It, therefore, calls for an alternative approach of performing reasoning outside the schema in a way that still supports semantic processing in a streaming environment. In this way, the alternative approach will still rely on the prior knowledge about the stream and/or ontology that provides the formal description of the stream including the related concepts and the relationships. In the following section, a review of the current and existing approaches that aim at providing reasoning support for data streams at a higher level of abstraction are discussed in relation to the IoT streaming data.

2.4.3.2 Enhanced C-SPARQL Stream Reasoning

In an attempt to complement and enhance the existing RSP technologies with adequate reasoning capabilities, many unified semantic reasoning approaches have continued to evolve. Most of the approaches attempt to layer the RSP systems with more expressive semantic rules that are sufficient to manage the data streams within the specific application. The following paragraph describes these categories of systems.

Most of the approaches (D'Aniello et al., 2018; Hoeksema & Kotoulas, 2011; Jajaga

& Ahmedi, 2017; Maarala et al., 2017) adopting the enhanced C-SPARQL reasoning are tied to specific domains such as Smart Cities or social media analysis. Most of these approaches rely on the use of C-SPARQL systems for accessibility, filtering or aggregation of raw data or RDF data streams and, later apply semantic rules or analytic algorithms to achieve some level of reasoning for inferring new knowledge about the data. One of the first categories of this approach is the Inductive stream reasoning system (Barbieri et al., 2010a) that combines C-SPARQL with statistical and machine learning technique to infer new knowledge. The approach aims at improving accuracy in the validation of a real-life scenario from social media analysis (Barbieri et al., 2009) by combining both methods of inductive and deductive reasoning. C-SPARQL is used to query RDF streams, while the deductive reasoning proposes an efficient technique to achieve materialization of the ontology entailment. The expressivity and soundness of the approach rely on the rule profile of OWL 2. Despite better performance in terms of latency, the approach does not support incremental classification and the Jena API. S4 (Hoeksema & Kotoulas, 2011) is another distributed high-performance reasoning system for RDF streams. In S4, C-SPARQL is enhanced with new operators to perform triple filtering, join, selection, projection, and aggregation to allow splitting of processing across many machines. The rule processing called *RuleProcessingPE* uses a key-value on all property of triples to perform the RDFS reasoning task on instances. The objective of S4 is centred around improving the performance throughput and scalability with no consideration for expressivity, classification or support for external API.

Recently, the focus has shifted into providing high-level reasoning by layering C-SPARQL systems with sets of rules. For example, (Jajaga & Ahmedi, 2017) is able to combine SWRL(Horrocks et al., 2004) with C-SPARQL filtering to achieve the C-SWRL system. The system was applied to water quality management (Jajaga et al., 2015) in an attempt to deduce the change in the water property. Though the approach was able to produce new knowledge, the system does not consider the classification or re-use of the knowledge produced. (D'Aniello et al., 2018) proposes a layered semantic reasoning-based architecture that combines the C-SPARQL with TrOWL(Thomas et al., 2010) to process sensor

streams generated within the context of Smart City in attempt to generate knowledge to support operational decisions by the government. In their implementation, C-SPARQL is applied for aggregation, filtering and searching of RDF streams. The reasoning layer is based on the method of syntactic approximation (Ren et al., 2010) which implemented in Java. The approach is aimed at reducing the reasoning complexity from 2NEXPTIME-Complete to PTIME-Complete. The expressiveness of the approach is defined around the ability to abstract high-level information from low-level data streams. Despite the improvement of reasoning over open time-annotated datasets, the effectiveness of the reasoning approach on run-time IoT streaming data is not guaranteed.

The expressivity and soundness of these approaches depend on the underlying rule language that is being adopted. All the reasoning tasks are performed on IoT and sensor data streams. There is no consideration for incremental classification by the approaches and only a few of them can support the Jena API for further or complementary IoT stream processing.

2.4.4 Ontology

Ontology refers to the formal specification of concepts and for knowledge representation(Gruber, 1995). It also facilitates knowledge sharing as well as merging and reusing of the represented knowledge. In general, Ontology models developed in OWL are mainly provided and exchanged as RDF graph or documents. This language formalism has made it easier for various semantic web groups (such as SSN incubator group, SWE, OGC, etc.) to standardize and facilitate interoperability of the sensor network while providing an approach to annotating sensor and IoT streams with a formal semantics that makes it more accessible.

2.4.4.1 Sensor Ontology and Domain Modelling

Sensor ontology models are designed specifically as either context-based/domain-specific models or as sensor network ontology models. The models usually forms the

basis for ontology extension in specific application or domain. In most application/domain-specific models (Gu et al., 2004; Psyllidis, 2015; Sandra Geisler, Sven Weber & Quix, 2011), the aspects of modelling and integrating sensor nodes with heterogeneous domain entities are the major focus. For example, in order to facilitate semantic context modelling, reasoning and knowledge sharing, the Context-based Ontology (Ali et al., 2017; Gu et al., 2004; Korpipää & Mäntyjärvi, 2003) was developed. Similarly, a number of specific domain-based ontology (Jajaga et al., 2015; Kim et al., 2008; Psyllidis, 2015; Sandra Geisler, Sven Weber & Quix, 2011) has continued to emerge in order to achieve better interaction among components in these domains. These Ontology models applied to various context or application domains appears to be parallel in terms of structure and fulfilling the requirements for interoperability, but the usage is often not domain-agnostic. In addition, very few of the ontology model consider quality dimensions of sensor stream as part of ontology model. Most Ontology models only focus on sensor node as context observing entity or physical device.

There have been a number of ontological models developed in the past years by researchers to represent sensors and sensor networks. Most of the sensor and sensor network Ontologies (Compton et al., 2012; Haller et al., 2018; Holger Neuhaus, 2009; Kim et al., 2008; Pease et al., 2002; Russomanno et al., 2005) are developed mainly to model sensor measurements, facilitate data fusion and provide a description for heterogeneous sensor nodes in a network environment. One of such ontology is used to model sensor nodes in adaptive wireless sensor network describing it as a concept with multiple components and relationships which is useful in modelling environmental and operating conditions (Avancha et al., 2004).

In observatory management, The OntoSensor (Russomanno et al., 2005) ontology is developed by using concepts from SUMO (Pease et al., 2002) to create a general knowledge base for the sensor in support for semantic query and inference. The ontology lacks clarity in the description of sensor measurement data and observation. A layered framework for the development of a universal sensor ontology model is developed by

(Eid et al., 2007) for the description of hierarchical sensor knowledge model, observational properties of transducers and support for domain-specific ontology models. The major focus of the ontology is on sensor data and measurements with little focus on the description of sensor nodes, systems and, procedures.

In the domain of coastal sensing, CSIRO ontology model (Holger Neuhaus, 2009) provides a description and metadata for sensors, procedures, groundings, process and measurement data. Its metadata does not include the description of sensor platforms. CESN ontology (Calder et al., 2010) describes physical sensors, properties, location and deployment in Coastal Environmental Sensor Networks. Similar to CSIRO ontology, it is inadequate in terms of describing the device/instrument and platform for sensor networks. The concept description is also limited to ten sensors instances and six individuals without consideration for the origin of the data. OOSTethys (Bermudez et al., 2009) is a similar ontology that provides a description of the ocean observation system. It is built with concepts that include the sensors, instrument, platforms and observation to support real-time data streaming. The model is observation-centric and perceives this as a system comprising of other systems or atomic processes. As a domain-dependent ontology, it does not consider the accuracy in the estimation of property values belonging to a feature of interest and fails to fully support querying.

As an attempt to encourage re-usability, the SSN Ontology (Compton et al., 2012) based on OWL2 is developed as an initial attempt to provide a comprehensive and reusable sensor ontology to provide a general description for sensors with sensors capabilities, accuracy, observations, performance as well as the operating and survival ranges. SSN ontology reused DOLCE-UltraLite[1] in the upper ontology and fully supports semantic query and data discovery in a sensor network environment. However, the system's perspective of the ontology does not include actuation procedures. The relatively recent version is the Modular SSN Ontology (Haller et al., 2018) that extends the previous SSN Ontology (Compton et al., 2012) to include the description of actuators, samplers

[1]http://www.loa.istc.cnr.it/ontologies/DUL.owl(Accessed 29/11/2020)

and observations including the sensor concept within its vocabulary. Developed as a lightweight Ontology, the core module contains SOSA (Sensor, Observation, Sampler, and Actuator) vocabulary[2] along with that of SSN. The Modular SSN Ontology was developed to support a wide range of applications including data-driven ontology engineering and Internet of Things applications.

In a related domain of IoT environment, the IoT-Lite ontology (Bermudez-Edo et al., 2016) was proposed with a motivation to reduce processing time. Built as an extension of the semantic sensor network (SSN) ontology, it provides a description for major IoT concepts allowing interoperability and discovery of sensory data in heterogeneous IoT platforms. The lightweight semantic of the ontology provides minimum concepts and relationship descriptions that do not include an aspect of the sensor data quality and support for evolution.

Reusing the existing ontology tremendously reduce cost(time/effort), resolves unnecessary ambiguous concept interpretations and, eliminate repetitions of properties and notions. Therefore, adoption of the re-use process involves a number of activities that includes Ontology search, assessment and selection. The search process was conducted using a set of inclusion and exclusion criteria. The inclusion criteria consider only models that consist of concepts describing sensors and measurement with other related concepts. Exclusion criteria provide a means to reject ontology models that are non-extensible and not in conformance to the Semantic Web standards. All relevant ontology models identified through the search process are subjected to Ontology assessment. Among the candidate Ontology models that were subjected to assessment includes SmartOntoSensor (Ali et al., 2017), OntoSensor (Russomanno et al., 2005), Sensor Node Ontology (Avancha et al., 2004), CSIRO (Holger Neuhaus, 2009),CESN ontology (Calder et al., 2010), SSN/SOSA ontology (Haller et al., 2018), Time ontology (Zhou & Fikes, 2002) and OOSTethys (Bermudez et al., 2009). The approach to Ontology assessment of the selected models involves a thorough inspection of the models at the

[2]https://www.w3.org/ns/sosa/ (Accessed 29/11/2020)

granularity levels of the objects and data properties to understands the suitability of the model. The result of the assessment activity is compared with the selected criteria for the target IoT and streaming data quality ontology during the Ontology Comparison activity. The specific requirements for the target Ontology (SmartSUM) include light-weight support, quality, re-usability, structure clarity and deployment cost. The most suitable ontology models based on the specified requirements are the Time ontology (Zhou & Fikes, 2002) for modelling temporal entity, and the SSN/SOSA ontology (Haller et al., 2018) based on its lightweight support and explicit definition of actuation and sampling procedures. The lightweight support (Poli et al., 2010) is described as one that provides a simple formalization of concept that is sufficient to manage the task under consideration. The selected ontology models are integrated into a target ontology and subjected to Ontology re-engineering using the Web Ontology Language (OWL) and protege 4.3.0(build 304) modelling tool. Despite the availability of various sensor and related domain ontology models within the semantic web community, it is essential to consider a more generic and lightweight ontology model that will be suitable for IoT stream quality management and efficient stream processing. The lightweight feature is expected to fully express the IoT domain and data quality concepts unambiguously. Therefore, providing a more expressive and reusable model. Similarly, the efficient stream processing feature must be able to support IoT streaming environment with an incremental updates.

2.5 Semantic IoT Stream Validation Requirements

It is common for different scenarios of stream applications to have different requirements because of stream characteristics (such as input rate, volume, or bursts), resource constraints, and the nature and complexity of rules. It is on this note that the essential requirements for a semantic-based validation system for IoT streaming data are proposed. Many of the requirements identified by this work are based on the general requirements earlier proposed by (Stonebraker et al., 2005) for stream Processing systems and (Margara et al., 2014) which detailed the requirements for stream reasoning

systems. These requirements are considered in addition to inference support which applies to semantic streams.

1. **Time Consideration *(R1)***

   The requirement describes one of the essential characteristics of IoT streaming data as being known to be dynamic, which mean they can change over time. A stream processing system will require putting this into consideration especially when it is integrated as part of applications such that the applications consuming the data can further process and responds in a timely manner. As an example, web-based controlling application responsible for the control of smart home temperature should be able to evaluate changes in sensor readings with timestamps, produced from various sources to validate the true reading of the actual room temperature in a timely manner. To achieve this, it means the streaming data from the sensor must be annotated with the individual timestamps(2014) and processed using a time-aware processing model that can define the relationships among these data.

2. **Data Integration *(R2)***

   In IoT applications, it is necessary to allow data from different external sources to be combined during information processing and knowledge extraction. For instance, it is possible to combine data from various sensors including the streaming data produced by same sensors to have detail knowledge of the environment. There has been continued recommendations for possible ways of integrating the sensor data from other sources(Gyrard, 2013; Psyllidis, 2015) to achieve interoperability. This justifies why the use of ontology modelling approach and RDF serialisation methods for sensor streaming data can be considered as part of the options in satisfying this requirement.

3. **Data-Driven processing *(R3)***

   Data-driven systems are normally constrained by data rather than the human experience or intuition. Most applications relying on IoT streaming data are required to be able to handle data processing and produce the result almost

the same time as it is generated by IoT node or sensing device. In time-critical applications like the health monitoring system for patients with heart related disease, the system must be able to process the sensor data "on-the-fly" before it becomes outdated. This means that results are produced just as the data are received by the application. This type of applications or systems are also regarded as active systems (Stonebraker et al., 2005), while its Passive counterpart are dormant until they triggered by external entity before they can start processing. As such, it ensures that the latency[3] of such system is kept within minimum.

4. **Semantic Stream Querying *(R4)***

   Developing a realistic mechanism for querying sensor and other IoT streaming data in IoT-based application provides an approach to retrieving and computing data of interest in real-time. Semantic stream processing system should be able to interrogate the streaming data over a processing window without resulting in overhead for the system. In addition, adopting an efficient query mechanism in a dynamic environment is a way to manage the memory consumption and response time of streaming applications. This requirement is applicable to data stream that is processed at the lower granularity level of the streaming application to ensure continuous real time data retrieval, and avoid undue conflict with the user view of the streaming application.

5. **Inference Ability *(R5)***

   Expressiveness is a key issue in semantic web applications as it determine the inference capabilities of the used model. Most IoT-based reactive applications should be able to support models that can handle streaming data in an efficient manner as well as reasoning functionality. The essence of enhancing semantic data model with reasoning system is to enable inference support to produce new set of knowledge from raw sensor streaming data. The process of extracting knowledge can require operator that facilitate data computation and transformation.

[3]Latency means the time between the arrival of new sensor reading and the generation of output or inference result

Such situation can include aggregating the readings from multiple sensors to compute the average reading.

6. **Stream Quality Management (Missing data, Out-of-Range Data)** ***(R6)***

   In real-time streaming application scenarios, data does not need stored before they are processed. In such situations, it is not advisable to allow applications to wait endlessly in order to identify erroneous data point within streaming windows . Hence, such type of application must be able to contain mechanisms that are driven by window-based strategy for denying access to erroneous or imperfect data, subscribed by a data processing model until the session or processing window is expired. It is therefore important to consider a basic infrastructure that must make provision for management of imperfections relating to missing and out-of-range data. The rule specified for stream processing requirement by emphasized on strategy that allow built-in mechanism to guarantee a resilient system against imperfect data streams from the real-world streaming data.

7. **Historical Stream Management** ***(R7)***

   The direct access to static IoT data previously produced by sensors, including the schema with background knowledge can provide a better understanding of the behaviour of a particular sensor. Combining such knowledge with the run-time IoT streaming data will update the system with information on how it is interacting with the environment at each streaming window. The application of semantic IoT data model with an evolving feature can also improve the rate at which streaming data with the associated timestamps is directly stored or persisted as new individual and historical data on the background knowledge graph following an update. This can be used in the prediction of future sensor reading or behaviour. Therefore, the historical stream management can involve any activity or process that realises stream querying, update or materialization of previously produced data streams.

The requirements contained in the survey on stream reasoning (Margara et al., 2014) also reflect similar requirements as those that have been identified above. The

work defined the requirements to include: integration which refers to data integration; efficiency which related to the data-driven processing and low latency in this work; time management is similar to time consideration in our case; expressivity similar to Inference ability; uncertainty management that relates to stream quality management in the present context; historical data management corresponds to the historical sensor stream processing; and big data Management which is directly tied to the data integration requirement in this work. The other requirements such as distribution and quality of service specified by the literature are not within the scope of this research and it will be considered as part of the future research.

On the other part, the requirements are motivated by the 8 rules defined for stream processing by (Stonebraker et al., 2005). The first requirement *Rule 1: "Keep data Moving"* is associated with requirement (R3) in this section which considers window-based processing and latency requirement. *Rule 2: "query using SQL streams"*. This requires that streaming data must be retrievable before the elapse time using a compatible query language which is related to the requirement (R4). *Rule 3: "Handle stream Imperfections"* in part is similar to requirement (R6), which explains the stream quality management. Other specific rules such as *Rule 5: "Integrate stored and streaming data "* is closely related to requirements (R2 and R7) in this section. *Rule 8: "Process and Respond Instantaneously"* which corresponds to requirement (R1) and requirement (R3) ensures that streaming data are processed more quickly with reduced latency. The remaining rules that include *Rule 6: "Guarantee Data Safety"*, *"Rule 7: Partition and Scale Application Automatically"* and *"Rule 4: Generate Predictable Outcomes"* are similar to requirement related to distribution in (Margara et al., 2014), which are not considered to be within the scope of this research.

Table 2.2: Analysis of Semantic Stream Processing Systems

System Requirement / **RSP System**	R1	R2	R3	R4	R5	R6	R7
EP-SPARQL(Anicic et al., 2011)	✓	✓	✓	✓	✓		✓
C-SPARQL (Barbieri & Milano, 2014)	✓	✓		✓			✓
Streaming SPARQL(Bolles et al., 2008)	✓	✓		✓			
$SPARQL_{stream}$(Calbimonte et al., 2010)	✓	✓		✓			
CQEL(Le-Phuoc et al., 2011)	✓	✓	✓	✓			✓
Instans(Rinne et al., 2012a)		✓	✓	✓		✓	
Sparkwave(Komazec & Cerri, 2012)	✓	✓	✓		✓		
IMaRS (Barbieri et al., 2010b; Dell'Aglio & Della Valle, 2014)	✓	✓			✓		
C-ASP (Pham et al., 2019)	✓	✓	✓	✓			✓

2.6 Analysing Requirements in RDF Stream Processing Systems

To provide an appropriate semantic stream validation system that supports the IoT streaming data for the quality validation process, analysis of the existing semantic stream processing systems are being conducted based on the requirement defined in section 2.5. The summary of the analysis is shown in Table 2.2.

Apart from C-ASP that is based on Answer Set Programming system (Gebser et al., 2013) , most of the semantic stream processing systems (Anicic et al., 2011; Barbieri et al., 2009; Bolles et al., 2008; Calbimonte et al., 2010; Komazec & Cerri, 2012) extends the native SPARQL query language, originally used for static RDF data processing. These systems introduced new operators or grammar that can support the temporal characteristics of semantic data streams. Among all the available semantic stream processing systems, EP-SPARQL, Sparkwave, C-SPARQL and INSTANS supports multiple streams query.

Time dimension (*R1*) is exhibited by all the systems but with little differences. C-SPARQL, C-ASP, $SPARQL_{STREAM}$, and CQELS uses a single point timestamp whereas, EP-SPARQL and IMaRS represents the time as an interval of timestamps. The only exemption to these timestamp dimension is INSTANS that relies on implicit schema definition.

As data integration (*R2*) remains the vital benefit of semantic web technology, all the existing stream processing systems can achieve this requirement irrespective of the semantic processing techniques adopted by each of the systems.

Majority of event-driven systems are known to be strictly bound by data, not all semantic stream processing system is driven by data (*R3*). For example, C-SPARQL,

$SPARQL_{STREAM}$, IMaRS and Streaming are controlled by model while EP-SPARQL, C-ASP, INSTANS, Sparkwave and CQELS are event-based and data-driven systems.

Mechanisms for gaining access to data streams of interests and/or outputs from the real-time computation of streaming data through semantic query (*R4*) has been exhibited by the majority of the stream processing systems except in Sparkwave and IMarS. Both systems are enhanced by external reasoning systems and are used to complement other semantic stream processing systems. In other words, they are only used to support other stream processing systems as they cannot perform a semantic query on their own.

The ability to produce new knowledge from underlying semantic data streams or facts is facilitated through a process of semantic inference (*R5*), which can be vital to the effective application of the semantic processing. This process is what has been described as semantic reasoning. Apart from EP-SPARQL, Sparkwave and IMaRS that has shown support for inference with external reasoner, the remaining systems do not include the feature as part of the stream processing approach. EP-SPARQL and Sparkwave are only able to provide inference on a subset of the RDF schema, while IMaRS is only able to provide inference for transitive property between RDF classes or nodes.

The quality requirement (*R6*) of streaming data is not considered as part of most of the existing semantic processing systems for data streams. It was observed that only INSTANS can capture a little aspect of the quality problems which has to do with the inconsistent data schema. Hence, processing stream quality at the data layer is not considered as part of the approach.

The integration and storage of previously processed IoT data streams with real-time generated streaming data is a demonstration of historical data management (*R7*). Previous Semantic streams processing systems such as C-SPARQL, EP-SPARQL and CQELS can combine the two types of data during processing as a way of fulfilling the requirement. There is still pending need to consider the requirements as part of the future semantic

streaming processing systems. Some aspects of the requirements may be sufficiently captured during the pre-processing stage or semantic serialisation of IoT streaming data.

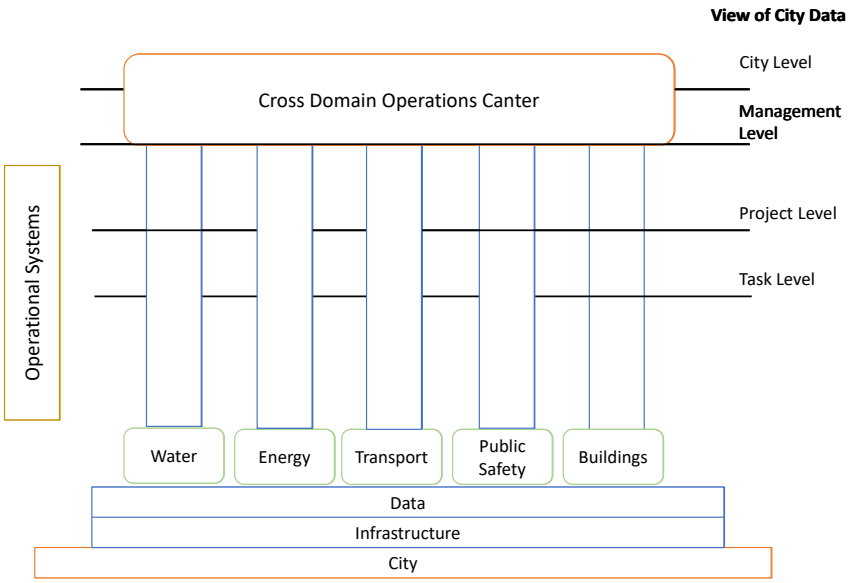

Figure 2.7: IBM Smart City Model

2.7 Smart City Model

In the past decade, a new revolution of city developments has been seen in most of the developed/developing nations, mostly driven by the Smart City models. Various models have been proposed by different researchers and organisations. These models are used to describe the abstract frameworks for all the different assets and aspects of a Smart City innovations (D'Aniello et al., 2018). Among all the models, IBM smart city model (Kehoe et al., 2011) and Smart city Layered framework (Kumar, 2015) are the most preferred models for data-driven decision. The models place control on technological infrastructures to facilitate better smarter cities and thus, supports smaller fragments of the smart city innovations including smart spaces as well as another related intelligent environment.

2.7.0.1 IBM Smart City Model

IBM smart city model proposed by (Kehoe et al., 2011) focuses on creating a smarter public environment. The model as depicted in figure 2.7 presents a formal technology-based infrastructure that is suitable for smarter city as well as providing support for city Governance in long-term planning activities. Compared to other Smart City models, IBM model is based on the concept of Decision Support Systems (DSS). The main objectives of the model on the Smart Cities vision are defined with two perspectives.

Firstly, it targets the quality of life of citizens and visitor of the Smart City with the purpose of adequate management, safety, sustainability, and good governance while incorporating people cultures and specific events. It also advocates for the need to focus on citizens by providing information and accessibility to smart city services in a convenient and usable manner. Consequently, the citizens and the government can benefit tremendously.

Secondly, the objective targets business growth and development as a means of building the economy of the city. In other words, digital innovation and commerce should form the basis for building the Smart City. Similarly, the challenges of city transport should be addressed by seeking to improve the cost of effectiveness and efficiency of the various city's transport methods.

2.7.0.2 Smart City Layered Model

The layered model (Kumar, 2015) is considered a building block of many layers involving various aspects of technologies enhancing data generation and gathering, data aggregation and analysis, and capability for optimal response. The model places emphasis on city operations as being data-driven and consists of five layers (see figure 2.8) essential for the success of the Smart City deployment. The connectivity layers comprise of various sensors/IoTs, Bluetooth or other connectivity infrastructure owned privately or publicly, that facilitates or allows cities to capture data across systems and responds

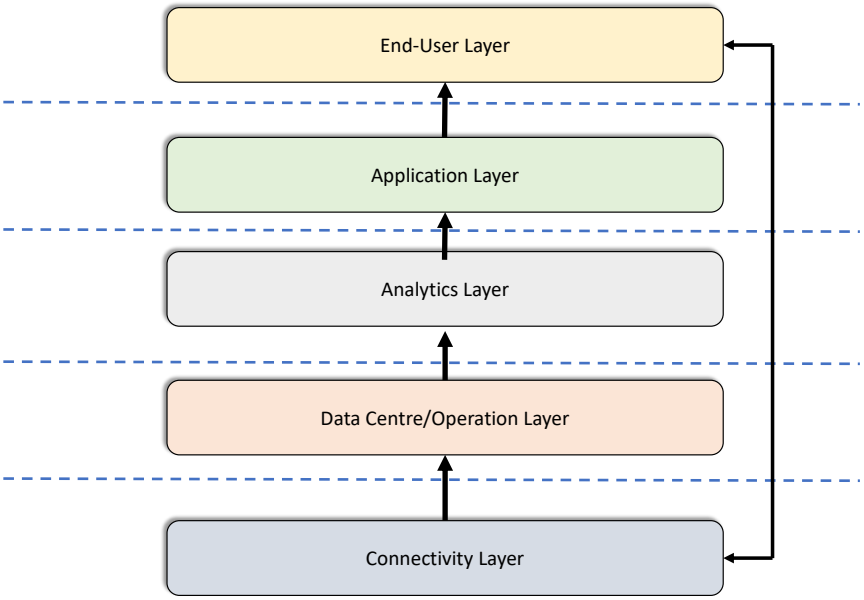

Figure 2.8: Smart City Layered Model

in an efficient manner. Based on the vast amount of data expected to be generated, Data centre/Operations Layer provides and manages both data storage and accessibility across various departments and applications. The Analytics Layer enables cities to exploits valuable insights from data collected from previous layers, thereby leveraging such as predictive analytics used by Smart Cities for optimal allocation of resources. Cities are able to implement various industry-specific and horizontal applications at the Application layer. The layer allows seamless integration with various services from entities, which can improve the overall Smart Cities efficiency. The end-user layer is the last layer of the building block that encompasses all resident, citizens, visitors and government of the cities. The layer captures data through smart devices including sensors and eventually subscribes to the outcomes of the Smart Cities initiatives.

Based on the emphasis from the two model to ensure the integration of various smart devices within the IoT ecosystem, effective use of these models can also promote the realisation of intelligent or Smart Spaces. The two models share similarities to the requirements that define the concepts of Smart Spaces, which is to support data-driven operations and decision support activities.

2.8 Conclusion

In this chapter, a survey of different but related quality problems associated with IoT streaming data has been presented. The problems are transient and require a validation approach (Kumar et al., 2006) that can identify and possibly eliminate them before they are exploited by other systems or applications. There has been an established indication that many of the statistical approaches lack completeness in modelling all types of problems and integrating these data for inference purposes.

The application of semantic technologies in solving many of the IoT or sensor streaming data quality issues is a step in the right direction as it has been successfully implemented in processing incremental RDF data related to event processing systems (Rinne et al., 2012a). In addition, serialised RDF data formats and RSP systems with the extension of SPARQL adequately supports the implementation of semantic stream processing systems. Most of the systems in this category are able to manage the temporal characteristics of the IoT data streams with pattern matching techniques.

There have been many contributions in the area of data stream processing (Garofalakis et al., 2016; Golab & Özsu, 2003; Hammad et al., 2004; Krämer & Seeger, 2004) in recent years. Many of these approaches consider relational and XML-based data streams that do not provide support for rich semantic elements (Bolles et al., 2008). Until now, contributions in the area of semantic stream processing systems have considered how IoT streaming data are applied at a higher level of context to support smart applications(D'Aniello et al., 2018; Jajaga & Ahmedi, 2017; Kamilaris et al., 2016; Maarala et al., 2017) and generate a new set of knowledge in the application domains. Furthermore, many of the RSP systems still require the support of formal specification of the data streams to enable them to perform effective stream processing task. This requirement is either done by using an existing ontology model for annotation of the data stream or embed such as external library within the RSP system for a unified system. Most of the

available RDF stream processing systems cannot still perform complex reasoning tasks such as handling the common-sense reasoning, recursion or preferences. For example, the attempt to provide reasoning functionality in C-SPARQL[4] only succeeded in allowing for RDFS entailment regime[5]. Hence, the ability of the stream processing systems to handle incomplete and inconsistent data streams while extracting knowledge for decision-making purposes in real-world applications is limited (Dell Aglio et al., 2017).

Ontology models so far developed for sensors and sensor networks can facilitate interoperability, and few of them are also able to represent sensor outputs using a generic description. A non-domain dependent lightweight ontology model like the Modular SSN (Haller et al., 2018) is able to provide a basis for the development of the ontology model for semantic stream applications. However, the ontology cannot be directly applied to the RDF stream processing systems due to variation in scope, lack of explicit concepts for stream quality and inconsistency in concept usage. This will be considered with the view of enhancing the Modular SSN ontology to support IoT Streaming data Validation in the subsequent chapter.

Developing the semantic validation approach using the combination of the ontology models, semantic reasoning and RDF stream processing technique is step in the right direction in overcoming the shortcoming of RSP in the aspect of reasoning. This will further provides a more effective use of ontology model on real time data rather than it well-known application on static data. Finally, the field of semantic processing of IoT streaming data is still at emerging stage and there are no standard benchmark for the semantic IoT stream processing systems currently proposed by researchers. Most of the current approaches only considered time-based evaluation without considerations for effectiveness of the approach. The various metrics have been put into consideration in this research and will be applied to a number of case studies for evaluation of the proposed approach.

[4]http://streamreasoning.org/resources/c-sparql
[5]https://www.w3.org/TR/rdf-schema/

Third Chapter

Lightweight and Evolving Semantic Model for IoT Domain and Streaming Data

---

3.1 Contribution

The contribution of this chapter consists of a new lightweight and evolving semantic model called SmartSUM ontology, which describes the related IoT resources including the streaming data with the quality metadata. Being lightweight allows unambiguous specification of concepts hierarchy and subsumption of relations between IoT nodes. SmartSUM ontology provides a semantic foundation for the machine understanding of the IoT stream data. SmartSUM ontology is built upon existing generic SSN/SOSA and Time Ontology with IoT stream quality annotation extension and it is able to provide the evolution features that support continuous updates and scalability of IoT resources including the semantic streams. The details of the features and processes are explained in this chapter.

3.2 Introduction

In an attempt to meet the second objective and to answer the first research question (**RQ1**): *How can the smart space domain and sensor network ontology support the in-*

cremental update of IoT streaming data caused by an addition of new data instance? the Smart Sensor Measurement-driven Ontology (SmartSUM) has been developed. SmartSUM differs from the counterpart sensor ontology models in that it is a light-weight model that leveraged the most recent modular SSN/SOSA ontology to provide descriptions for sensor streams quality dimensions and support popular RSP systems with continuous reasoning capabilities. As such, it enhances the RSP systems to simplify the process of stream quality validation in IoT streaming data at the lower level of context before IoT streaming data are being exploited by reactive services. Unlike similar sensor ontology models, SmartSUM is able to provide the common schema for the main actors of IoT environment and smart spaces including the temporal characteristic and quality dimensions of IoT data streams. The detail design and construction of SmartSUM ontology is discussed in section 3.3.

The second unique feature of SmartSUM ontology is the support run-time incremental updates resulting from addition of new data instances. The resulting ontology from updates describes the evolving functionality of the ontology model. The evolving functionality of the ontology is enhanced with Knowledge Graph (KG) embedding techniques for the semantic space so as to allow for the classification and relationship extraction of newly identified entities in the space. Adequate consideration for the time requirement of a semantic matching process during the evolutionary process is part of the choice considered for the KG embedding process. This is discussed in more detail in section 3.4.

3.3 Ontology Model Design

Providing metadata for heterogeneous sensors and the observations is no doubt one way of ensuring effective interoperability and accessibility to measurement data within Smart Spaces and sensor network environment. Achieving this task is facilitated by the development of SmartSUM Ontology. The SmartSUM model consists of a formal conceptualisation and specification of Smart Space components/resources and sensors

including the types, taxonomy, relationships, and metadata for sensor characteristics, performance, and data quality metrics. In addition, SmartSUM contains statements describing associations between smart space resources and sensor concepts as well as aspects of the operating principles, platforms, observations and measurements, and other related semantic contents. The main justification of the SmartSUM Ontology are given as follows:

1. Provide a semantic model for the description of essential features of Smart Spaces including physical sensors with their measurements thereby, enhancing IoT-based reactive applications to make effective data-driven decisions and actions;

2. Support IoT-based reactive applications with a semantic interface for discovery, semantic processing, integrating, and annotating heterogeneous IoT streaming data produced by sensors including the timestamps;

3. Support the process of semantic reasoning and inference for the realisation of Ontology Evolution during the addition of new ontology instances and concepts.

In order to achieve the semantic model sufficient for the description of sensor and the major smart space resources, the design considered an extension of a number of existing generic and related ontological models as the base models. This allows for the unambiguous description of IoT and smart space domain concepts with the relationships. Specifically, the semantic model identified the essential smart space resources (i.e. sensor, actuator, measurement data, controllers and hub) in the class hierarchy description with the class relationships to related domain concepts.

Based on the time constraints and dynamic nature of data within the IoT environment (Tu et al., 2020), the second contribution of the model is able to capture the temporal characteristics of the measurement data by supporting discovery and run-time annotation of sensor streaming data using the serialised RDF data format. Therefore, providing support for improved expressivity and interoperability among heterogeneous IoT entities.

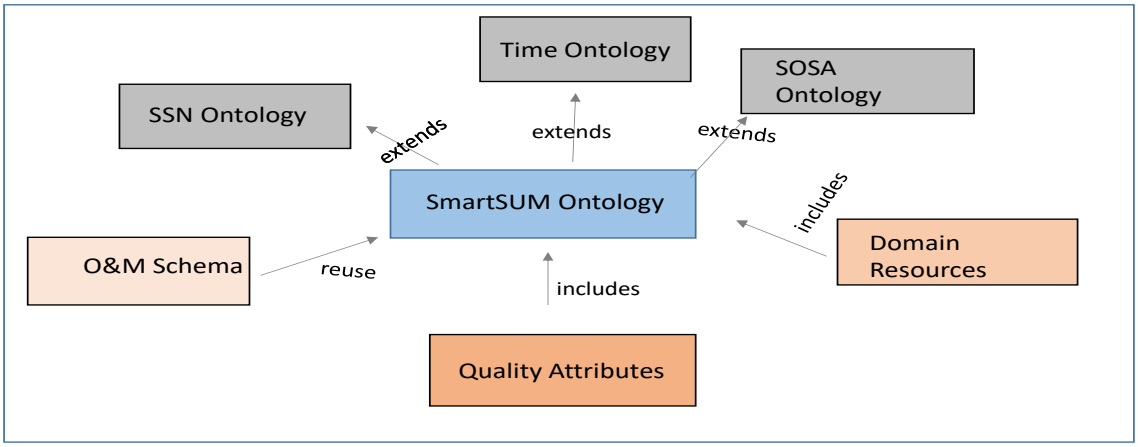

Figure 3.1: Abstract Ontology Reuse Framework

The third feature involving the ontology evolution supports the incremental storage the streaming data as historical data on the semantic model as a sequence of RDF data format rather than OWL format. The feature is based on the non-heuristic classification method that involves RDF graph embedding with semantic matching to manage the application of a change in the dynamic IoT environment.

Since the SmartSUM ontology will be required for the annotation IoT streaming data at a lower level of granularity, its design pattern follows the scenario on "Ontological Resource Re-use" process as contained in the Neon Methodology (Suárez-Figueroa et al., 2012) for Ontological development. The pattern involves activities including ontology assessment, comparison, and selection in the ontology reuse phase. It is suitable for the description of heterogeneous distributed objects in the real world. This is often realized by shared and common theories defined on a certain domain. This will not only help in human understanding but also helps the machine to communicate easily by exchanging the semantics of the data or object instead of the syntax. This Ontological re-use activities are later complemented with the Ontology re-engineering process to achieve the novel evolving semantic model.

3.3.1 SmartSUM Design Process

The design and construction of SmartSUM ontology have been achieved with the reuse model which involve the use of existing and related ontological model as the upper ontology for concepts description. Specifically, the design pattern is similar to scenario 4 of Neon methodology, which clearly explains the process of Re-use and Re-engineering of Ontological concepts.

3.3.2 Ontology Reuse and Re-engineering for SmartSUM Construction

The combined SSN/SOSA with the Time ontology formed the basis for the Ontology re-engineering and reuse process during the construction of SmartSUM. Figure 3.1 shows the abstract framework of the ontological resources that form part of the reuse and re-engineering process. In this work, the ontology re-engineering process is used to facilitate the process involved with the introduction of new features representing the class concepts and properties to the base ontology model. During the Ontology re-use process, all the related base ontological models are linked together through a method of ontology alignment. The application of both ontology re-engineering and re-use process therefore re-innovated and created the new semantic model called SmartSUM.

The resulting new Ontology model consists of six separate logical modules organised into upper Ontology models and domain concepts. The upper Ontology consists the Time Ontology (Zhou & Fikes, 2002), SSN ontology (Compton et al., 2012) and SOSA Ontology (Janowicz et al., 2019), while the domain concepts are the specific domain related resources. This includes the details of IoT streaming data quality attributes, sensor steams and measurement data specification formally adapted from O&M schema and specific objects or resources in smart spaces.

The SmartSUM hierarchical class diagram in Figure 3.2 represents the taxonomy of

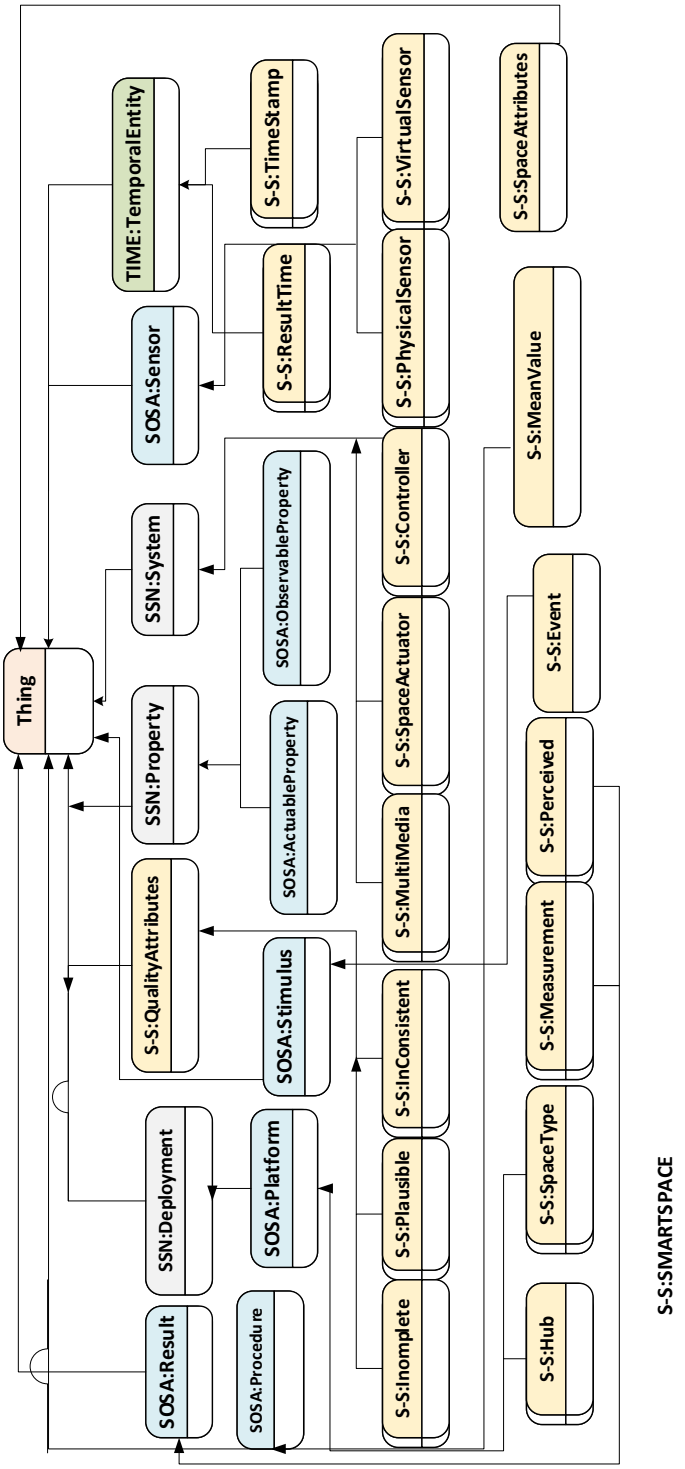

Figure 3.2: Abstract SmartSUM Model

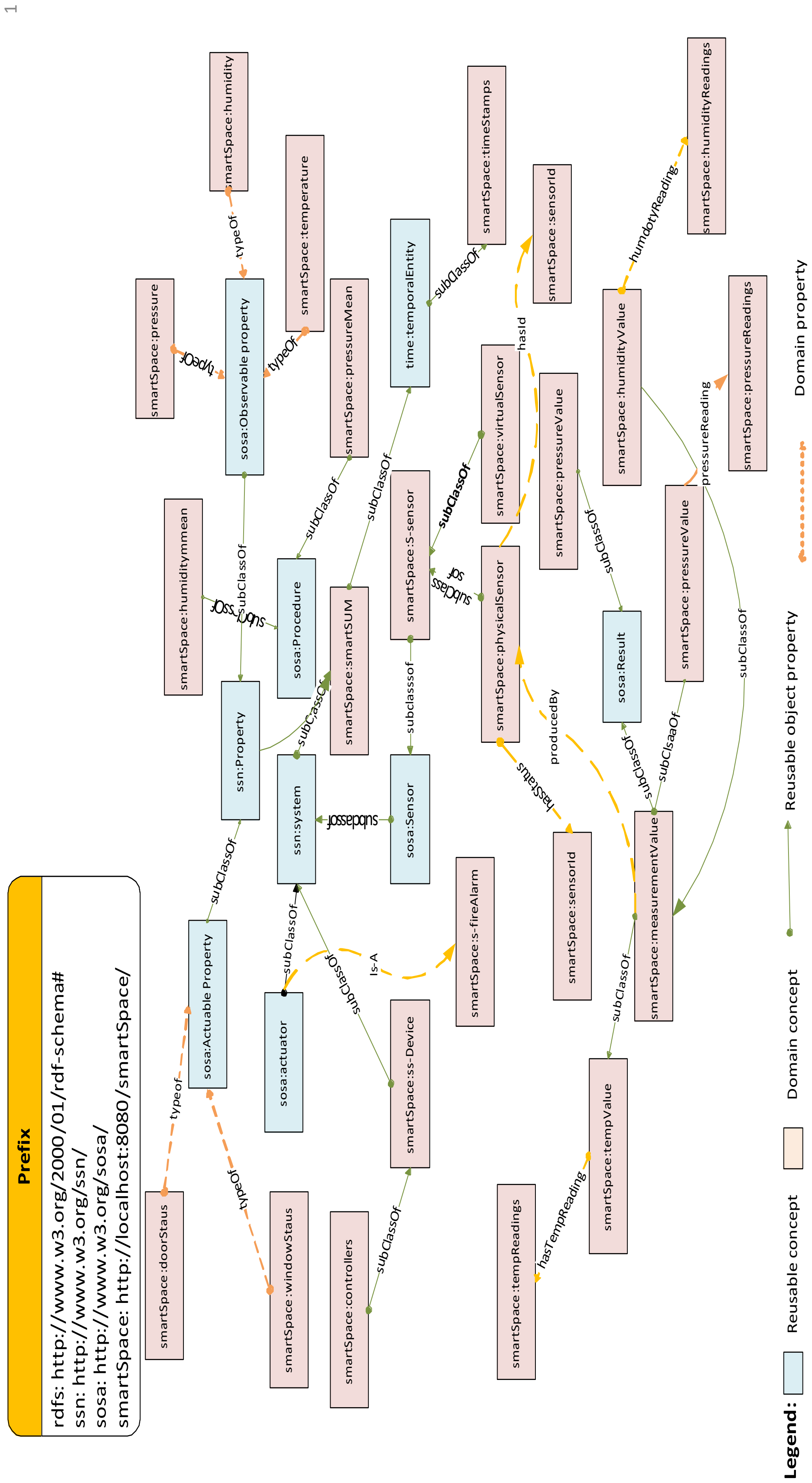

Figure 3.3: Snippet of Properties in SmartSUM Model

concepts that describes the major concepts involved in the construction of SmartSUM ontology. The ontology is mainly developed from concepts that are specific to IoT streaming data and smart environment/spaces. The hierarchical arrangement of the class or concepts is determined by the relationship that exists between them, which allow a concept to be either a superclass or subclass. The major concepts from the upper ontological model are used directly and extended where applicable to avoid unnecessary ambiguity among concepts and to achieve a reusable lightweight model. In addition, concepts with the same semantics from the reused upper ontologies are declared as equivalent (e.g. $SSN:Output \equiv SOSA:Result$). New domain-related concepts not found within the upper ontology models are explicitly created as either a superclass or subclass in the concept hierarchy.

SmartSUM Ontology currently contains a total of 144 new concepts with 59 object properties and 42 data properties. Each concept is unique and is defined according to the specific requirement of the semantic validation approach and the domain of application. A snippet of the object properties and data properties with domain concepts are shown in Figure 3.3. The namespace adopted by SmartSUM ontology model is known as *SMARTSPACE*. For instance, to re-engineer and reuse the upper ontology for the ontology construction, SmartSUM extends the SOSA ontology by organising the concept as $SMARTSPACE:Hub \subseteq SOSA:Platform$. This means SMARTSPACE:Hub is a kind of Platform for hosting other sensors, actuators, controllers and other smart space systems. The SMARTSPACE:DoorActuator, SMARTSPACE:VoiceRecognition and SMARTSPACE:Controller are all subclass of $SSN:System$ to enable proper identification of what classifies as system and differentiate from other concepts. Furthermore, the SOSA:Sensor enables the definition of any object that can respond to stimulus. A sensor detects inputs and produces certain outputs. Therefore, both SMARTSPACE: PhysicalSensor and SMARTSPACE:VirtualSensor are a subclass of SOSA:Sensor in the concept taxonomy. The SMARTSPACE:PhysicalSensor represents the hardware-based sensors (such as temperature sensor, camera, pressure sensor, etc.) while SMARTSPACE: VirtualSensor concept is for non hardware-based or software-based sensor (such as

crowd, feelings) captured within the space. The sensor hierarchy can be further extended to allow more detail specification of sensor type in a domain of application. As an example, SOSA: FeatureOfInterest is used as a superclass for concepts such as $SMARTSPACE:Temperature, SMARTSPACE:CO_2, SMARTSPACE:NO_2$. The $SMARTSPACE:SensorReading \subseteq SOSA:ObservableProperty$ is a concept that describes various sensor readings from various properties such as temperature, pressure, humidity, Benzene, etc. Each of the readings is associated with $SMARTSPACE:Timestamp \subseteq TIME:TemporalEntity$. Likewise the SMARTSPACE:ResultTime is a subclass of TIME:TemporalEntity and it represents the time that the result of the IoT streaming data validation process is completed. Both SMARTSPACE:Disturbance and SMARTSPACE: Season is subclasses of SSN:Stimulus as they model a set of events that can trigger the sensor readings and influence the observable properties within the space. Another major class concept that was introduced to the ontology model is SMARTSPACE: QualityAttribute that describes the major quality dimensions attributed to IoT streaming data.

Finally, the use of class axioms provides means of binding classes to other axioms in order to form Object or individual instance in a semantic and non-conflicting manner. For instance, a disjoint axiom is declared on classes that belong to the same hierarchy to restrict separate individuals' classification such that SMARTSPACE:VirtualSensor and SMARTSPACE:PhysicalSensor are disjointed to ensure that an individual instance can only belong to a maximum of one Class at a time and not more.

Comparing SmartSUM ontology with the base ontology models involved in the reuse process, it is established that none of the base ontology models considered data uncertainty and data quality dimensions as part of the concept description. For example, SSN and SOSA ontology only identify the output of observation from a particular measurement as *ssn:Result* without a specification for the quality attributes of the result. SmartSUM ontology does not only specify the output of measurements but also different the types of results that can be produced by sensors while including the various data

quality dimensions. Similarly, the SSN ontology have integrated the Time ontology as part of the modular SSN/SOSA ontology. The current re-use of the Time ontology only specify the generic time instant of the measurement. In the SmartSUM ontology model, the Time ontology has been updated to include the season that a particular result or measurement is recorded.

3.4 Evolving functionality of Domain Sensor Streaming Data Ontology

Domain Ontology construction often involves manual specification of concepts and properties resulting in huge taxonomy of classes. It is ideal to consider that this taxonomy of classes are subject to a process of evolution especially in a very dynamic environment such as smart spaces (e.g. smart home) where IoT sensors or resources can join or leave the environment at regular interval. The evolution process can happen through the application of change involving addition or removal of specific resource/IoT device from the environment. The manual method (involving experts) of updating the base ontology to reflect the new changes has been the only reasonable approach. However, this method may become very expensive and complex especially with a large knowledge graph comprising various classes and properties.

This section proposes an automatic approach for annotating and classifying IoT resource new to smart space environment at the point of device profile registration. The approach provides meta-data for IoT resources including the streaming data, which will be complemented by non-heuristic classification and with support of graph embedding process. The technique of non-heuristic classification has been investigated in semantic web services (Corella & Castells, 2006). The non-heuristic classification is able to perform the assignment of specific IoT resource or device that recently joined the smart environment, to the taxonomy of the domain ontology. The IoT resource classification with a non-heuristic approach is achieved with the support of one or more

base ontological models within the ontology registry or Knowledge base.

3.4.1 Significance of SmartSUM Ontology Evolution

In the past few decades, considerable efforts have been put into the development of various ontology models to describe and support various domains for sensor applications. Most of these ontology models can encapsulate the domain knowledge within its structure and yet fails to evolve with it rapidly. Nevertheless, it is evident that the domain knowledge keeps evolving continuously in an open dynamic environment (Fensel, 2001). One of such dynamic environment is the open pervasive sensing environment like the smart spaces. In smart spaces, the evolution from ontology change can be caused by the *Environment, users* which in this case can either be the ontology engineers or system applications (Stojanovic, 2004), and *IoT Resource Classification.*

1. *Environment :* This confirms the environment of the system or smart space can change thereby altering the initial ontology conceptualisations which the system was built. For instance, an ontology may have been built to support a Smart home environment, trying to use the same to support similar processes in a smart manufacturing environment will require certain change process.

2. *Users :* The software or application requirement of a system may require certain changes once it has been deployed. As an example, adding a new type of sensor to support the task of IoT-based reactive application will lead to new system requirements.

3. *IoT Resource Classification :* Resource classification is the key strategy for ontology construction. IoT and smart object classification must always be accompanied with detail specifications of the sensor streams to enable the semantic understanding of the space whenever the data change.

An ontology is usually developed using an iterative method for it to be useful by applications. Once it has been deployed for use, it is normal for its underlying knowledge to change over time. For instance, the use of an ontology to process sensor information

and measurements in critical systems or reactive services (e.g. fraud detection, fire and safety, etc.) often comes with the need for further or continuous updates to the knowledge base to reflect new findings or support the recent changes to related applications. The cost of developing a new ontology or modify the existing one to reflect the changes using the traditional methods is usually very expensive in terms of time and efforts by the ontology engineers. This is because the ontology engineer cannot track and understand all the effects of changes throughout the ontology. The evolving feature of SmartSUM therefore facilitate ontology evolution process in order to accommodate all the changes and maintain consistency of classes and relations.

3.4.2 RDF Graph Embedding with Semantic Matching Model for Ontology Evolution

Process embedding was first proposed for sophisticated real-time control systems and reactive systems that involve graph structures (Kaelbling et al., 1987). The SmartSUM ontology can alternatively be implemented as a Knowledge Graph (KG) while being processed as RDF triple statements, which as a result provides the semantic descriptions of the IoT objects, sensors and measurements with timestamps. The Knowledge graph is embedded to support the automatic classification and relationship extraction to describe the IoT domain semantic model. One of the major downstream application of Knowledge graph applications is considered in terms of the embedding process during entity resolution (Wang et al., 2017). The use of KG embedding models provides an automatic method for transforming data into an existing feature that is based on the underlying knowledge and principles. Here, the proposed concept of KG embedding is used to enhance the scalability of the SmartSUM ontology in terms of its evolving characteristics and related reactive IoT applications. In this regards, the process of knowledge graph embedding is designed to enhance the automatic addition of new IoT entity and streaming data into the base ontology.

The techniques for Knowledge Graph embedding is categorised into Translational

Distance Models (TDM) and Semantic Matching Models (SMM). A reasonable approach to embed the RDF graph with IoT vocabulary into a dynamic space will require the adoption of the techniques for Semantic Matching models. It will allow the matching and automatic classification of new IoT concept or resource for proper integration with the base RDF Graph (SmartSUM Ontology). Unlike the *Translational Distance Model*, Semantic Matching model can facilitate the process of ontology evolution by adopting the semantic-based scoring functions to determine the similarity of IoT resources and sensor streaming data. The technique is also able to compute the plausibility of RDF statements or facts through the latent semantics matching of the statements(or entities) and the relationships contained in the RDF graph. A fact is often described as triple $\mathbb{F}^{+} = (\mathbb{s}, \mathbb{r}, \mathbb{o})$ with the relationship. The Semantic Matching model adopts a special class of model called ANALOGY (Liu et al., 2017) to achieve this task. ANALOGY is a bilinear model that extends RESCAL (Nickel et al., 2011), which determines the Latent Semantics by associating each entity with a vector. ANALOGY model analogical relations *r* and concepts *s, o* such as *"TemperatureReading-**is-a**-SSN:Output"* ***as*** *"PressureReading-**is-a**-SOSA:Result"*, by using a bilinear scoring function as follows:

$$f_r(s, o) = s^T M_r o$$

where $s, o \in \mathbb{R}^{\mathbb{d}}$ are the vector embedding for entities and $M_r \in \mathbb{R}^{\mathbb{d} \times \mathbb{d}}$ is the linear map associated with relation. To further model the analogical relations, it is necessary to determine the normality and the cummutativity of the linear map of the relation as follows:

$$Normality: M_r {M_r}^T = {M_r}^T M_r, \forall r \in \mathbb{R}$$

$$Commutativity: M_r M_{r'} = M_{r'} M_r, \forall r, r' \in \mathbb{R}$$

A multi-dimensional property of linear maps of two relations is usually considered as a directed path with similar Subject nodes and object nodes that forms a "Compositional Equivalence". These equivalent compositions of the linear paths are described as the Commutativity Property of the linear maps, which is a necessary requirement for defining the analogical structure for the evolving model.

3.4.3 Consistency Model for ontology Evolution

The use of semantic reasoner is an approach to maintaining consistency in Ontology models. The Pellet reasoner is considered as an appropriate choice to manage consistency of newly added or defined ontological concepts or resources. The choice is due to some reasons that include the fact that Pellet supports incremental classification (addition and removal of concepts) and ABox reasoning with high expressivity. Compared to other ontology reasoners, the pellet is available as open-source, compatible with all platforms, provide native rule support and work well with both OWL and Jena APIs. It supports Java as an implementation language and can significantly improve reasoning with its soundness and completeness.

Furthermore, the consistency model is currently defined as IoT streaming data persistence during the ontology evolution process. The ontology evolution based on IoT streaming data results from a frequent update of the persisted data or addition of new data instance. Here, IoT streaming data is described as RDF triple (containing the readings and its semantics) with a timestamp. The goal of the evolving property of SmartSUM ontology is to ensure that the application of changes must result in the ontology conformity to consistency without any loss of data.

Definition 4.1: A single stream ST_r within a particular time window T_w is consistent *iff* it maintains specified constraints for each individual time window T_w within each timestamp. Since the streams consistency for a particular ontology is defined, the set constraints will then depends on the underlying semantic rules. In the remaining part of this section, It is worth noting that most of the definition of concepts provided in this section is similar to (Stojanovic, 2004). Then, the stream consistency model $M_(STr)$ is defined as:

$$M_{STr} = C_{Rules} \cup S_{constraints} \cup U_{constraints}$$

Where C_{Rules} are the consistency rules of the model, $S_{constraints}$ are the soft-constraints and the $U_{constraints}$ are the user-defined constraints. These constraints are defined in relation to sensor streams. The same situation holds for the consistency of ontology

concepts.

The evolving feature of SmartSUM ontology is perceived as elementary changes that represent simple, fine-grained changes caused by frequent automatic update performed on the ontology instance. The propagation of this change is closely monitored to maintain and preserve the consistency of the ontology throughout the process. The meta-change transformation of the ontology is best described as *AddConcept* or *AddInstanceOf* depending on the complexity of the change (higher or lower). For example, each concept representing a property or phenomena is automatically populated with its instance without conflict. Suppose a single Smart space ontology subject to changes is represented as *SpaceOnt*:

Definition 4.2: The change Δ between ontologies is a direct mapping between $SpaceOnt_1$ (initial ontology) and $SpaceOnt_2$ (resulting ontology) such that

$$SpaceOnt_2 = \Delta SpaceOnt_1$$

It is important to note that in the definition $SpaceOnt_2$ is the changed ontology. This change is tracked along with the individual sensor reading. A change is additive when the entities (i.e. concepts or instance) of the resulting ontology are added without altering the existing one. In the present work, data instance can be added to the existing ontology while a new set of sensor reading is being produced. Likewise, new concepts can be added automatically whenever a new sensor node is detected in the space.

3.5 Conclusion

In general, semantic models are not meant to be used as a main or overall solution to a problem. They only usually form part of a target solution, which are often expected to be transparent to the end-user applications. Sometimes used as semantic annotation models, it should be offered with effective approaches, API and tools to process the semantics during the extraction of actionable information from raw IoT streaming data.

The present-day non-semantic stream processing systems including the DSMS and CEPs enhances processing of data and event-driven applications without the use of ontology or knowledge graph models which is required to define common metadata for streaming data. Hence, they are unable to perform the reasoning task on sensor streaming data. The strength of the semantic modelling provides a solid foundation for the present study to provide more structured metadata for sensors and IoT streaming data including domain concepts for easy integration and interoperability of various heterogeneous streaming data and resources. This chapter provides a new semantic model for the Smart space and IoT domain to support the description of domain concept including the IoT/Sensor streaming data. The SmartSUM Ontology can be further adapted and re-used in the related domain of application.

The Semantic Sensor Network Ontology is known to be the most widely adopted upper ontology for many sensor applications but its potential to handle the evolving feature that supports sensor streaming data is not yet evident. Furthermore, efforts have been committed to the semantic modelling and reasoning for IoT data streams within various smart cities projects (D'Aniello et al., 2018; Gu et al., 2004; Gyrard & Serrano, 2016; Jajaga & Ahmedi, 2017). However, what is lacking is a generic unified framework that integrates the semantic model with deductive with temporal capabilities to facilitates the run-time quality validation of IoT streaming data. Besides, the method of IoT Streaming data validation is still open for further research and remains unexplored.

The following chapter provides an evaluation method for the SmartSUM ontology through the application-based evaluation approach (Wang et al., 2014), where SmartSUM will integrate with the validation framework that largely influences outputs of the semantic validation process. Details of the proposed unified semantic validation framework for IoT streaming data will be presented in the next chapter.

Fourth Chapter

Semantic Validation Approach and Unified Framework- ***SISDaV***

4.1 Contribution

The contributions of this chapter include; specification of requirements for the Semantic validation of IoT streaming data quality; the development of a unified semantic approach with its supporting framework for IoT streaming data quality validation. In an attempt to achieve the validation requirements, the other key contributions include; the Semantic time-aware forward inference rules; the RDF streaming data selection; and the continuous semantic reasoning.

4.2 Introduction

The previous chapter provides the semantic modelling and description for IoT streaming data and smart spaces with the use of web ontology language. While the model provides a generic method for IoT streaming data annotation, it is not yet sufficient for complete IoT streaming data validation. In an attempt to satisfy the third and fourth objectives and, in view of answering the research questions (**RQ2.** and **RQ3.**), a **S**emantic **I**oT **S**treaming **Da**ta **V**alidation (***SISDaV***) approach and its resultant framework is developed. The approach adopts methods of semantic modelling and reasoning to achieve the quality validation process against plausibility, inconsistency and incompleteness

in IoT streaming data. The semantic modelling aspect of the approach is based on SmartSUM model developed in the previous chapter. Section 4.3 provides the details of the semantic IoT streaming data validation approach while the approach integration resulting into the generic framework is presented in section 4.4. section 4.5 presents the chapter conclusion.

4.3 Semantic IoT Streaming Data Validation Approach

As semantic modelling of the IoT data streams is often realised with the use of the RDF data model in conjunction with other semantic processing systems, most semantic stream processing systems based on RDF stream processing are still not fully explored (Zhang et al., 2012) in the aspect of reasoning on RDF streams. Also, the quality of the IoT streaming data during context reasoning is still subject to question especially in terms of inconsistent, plausibility or missing sensor and related IoT readings.

This section describes the major features of the semantic approach for validating the IoT streaming data based on the requirements specified in section 2.5. The framework (SISDaV) based on semantic approach is applied to the processing of the raw IoT streaming data to enable the accomplishment of the full quality validation process of IoT streaming data. Besides, it provides run-time data representation, integration, and querying over large IoT streaming data. Instead of directly applying one of the existing ontology/RDFS reasoners, the framework enhanced the Jena rule including the reasoning subsystem with some temporal characteristics to support IoT streaming data processing with a rule-based approach. The distinguishing design decisions and features of SISDaV are described in detail in the subsequent sub-sections.

4.3.1 Data Transformation

To ensure proper interoperability and integration of the IoT streaming data, it is important to consider the adoption of a unifying model such as the one for data representation that will facilitate these objectives. In this way, it will be possible to query and reason across the semantic streaming data produced by the various heterogeneous IoT nodes.

The RSP systems described in chapter three uses the RDF data model for data streams processing. They define RDF stream as an individual ordered sequence of RDF triples with an associated time element.

The raw IoT streaming data (usually infinite sequence of data) represented as d^{str} can be perceived as physical streaming data (also called internal streaming data) having a validity modelled as a time interval. This is meant to be transformed into an equivalent semantic or logical streaming data for proper quality validation to be possible. The validity associated with physical streaming data refers to the application time and not the system time. It is based on the fact that the modelling of time element from raw IoT streaming data will be based on a specific application. The time element of raw IoT streaming data is considered expired, once it has no direct impact on subsequent processing result. The pre-processing approach of IoT streaming data borrows some aspects of the techniques used for continuous data streams queries (Krämer & Seeger, 2004), while adapting such to suit the semantic approach.

Let T be defined as discrete pair of time instant and order total, such that $\mathsf{T} = (\mathbb{T}, \leq)$ (Bettini et al., 1998). Let

$$\mathbb{I}_{Time} = \{[t_{start}, t_{end}) \in \mathbb{T} \times \mathbb{T} \,|\, t_{start} < t_{end}$$

be set of the partially closed time intervals.

Definition 5.1: A set of pair

$$S_{Physical} = (\mathbb{K}, \leq_t)$$

is considered a physical IoT streaming data if :

- $\mathbb{K}$ represents an infinite sequence of IoT streaming data with its associated timestamps within a time interval:

$$[t_{start}, t_{end}) \in \mathbb{I}$$

- Each IoT streaming data is associated with a specific streaming window

- $\leq_t$ is the order relation for which $\mathbb{K}$ usually ordered by timestamps.

Definition 5.2: $S_{Logical}$ is defined as a semantic streaming data containing a triple $(s, p, o) \in \mathbb{L}$ such that $\mathbb{L} \in \mathscr{K}^{\mathscr{G}}$ and, a point in time $t \in \mathbb{T}$ holding the following conditions:

$$S_{Logical} : \forall (\mathbb{L}, t) \in S_{Logical} * \nexists (L^1, t^1) \in S_{logical} \in \mathbb{L} = L^1 \wedge t = t^1$$

where $\mathscr{K}^{\mathscr{G}}$ is the domain Knowledge graph modelled as RDF graph. The transformation of raw IoT streaming data (also known as physical streaming data) into semantic streaming data follows the assumption that IoT nodes comprise of sensors and applications that produces streaming data with associated timestamps. It is generally believed that IoT streaming data are ordered by the corresponding timestamps. If IoT streaming data arrives at a stream quality validation system without a timestamp, the validation system can append the streaming data with the timestamp at their arrival using the internal system clock.

At the initial processing stage, the *start* timestamp is considered as the *start* of the physical stream with the corresponding *end* timestamp as ∞, based on the assumption that each d^{str} will remain valid throughout. In that regard, each d^{str} is directly mapped to $(d^{str}[t_{start}, \infty))$ in $S_{Physical}$, where t_{start} represents the timestamp associated with d^{str}. The implication is that the *Order* of d^{str} is well preserved in $S_{Physical}$. Therefore, the structure of $S_{Physical}$ is able to extends d^{str} with additional two timestamps that represents the *start* and *end* timestamps respectively.

In order to transform from Physical to Semantic streaming data, Let $S_{Physical} = (\mathbb{K}, \leq_t) \in \mathbb{S}_{Physical}$ be Physical streaming data. A transformation $\partial : \Re(\mathbb{S}_{Physical} \times \mathbb{I}) \longrightarrow S_{Logical}$ holds for a physical streaming data from $S_{Physical}$ to its semantic $S_{Logical}$ equivalence as follows:

$$\partial(\mathbb{K}) = \{(l, t, w) \in L \times T \times W | w = |\{(l, [t_{start}, t_{end})) \in \mathbb{K} | t \in [t_{start}, t_{end})\}|\}$$

.

All individual IoT streaming data $(d^{str}, [t_{start}, t_{end}) \in \mathbb{K}$ in form of physical streaming

data are converted into triple (*s,p.o*) patterns by establishing the relationships with other IoT components including related streaming data, using the underlying domain ontology. The interval time requires to be split into point of time (time instant) at lowest time granularity. This will provide an idea of all the exact time instants that the IoT streaming data is valid. Since each IoT streaming data or physical streaming data is produced on a window basis, it is ideal to consider the window size *w* as part of the transformation process. The transformation of the raw IoT streaming data into semantic or logical representation provides a foundation for semantic serialisation process.

The serialised formats of semantic IoT streaming data (as discussed in section 2.4.1) is accessible to the framework to achieve a complete data pre-processing requirement and a unified data representation. Subjecting the raw IoT streaming data to these serialisation options of the RDF data will allow the framework to carefully evaluate and suggest the best alternative for semantic modelling or representation of the IoT streaming data.

The semantic serialised format of IoT streaming data can be easily taken as input during query processing. In this regard, the selection and ordering of the semantic are more achievable. The semantic pre-processing approach to raw IoT streaming data that is currently adopted by the framework is Data-driven and therefore satisfies the requirement **R3** for semantic stream validation system.

4.3.2 Multiple Streaming Data Selection

Based on the characteristics of streaming data, it is practically impossible to store raw IoT streaming data entirely before processing them. One may be able to register a stream query that can process serialised IoT streaming data as they are produced. Among all the various semantic stream processing systems, the C-SPARQL grammar with the corresponding syntax is currently adopted for enhancement by the semantic validation approach. This is due to the availability of its test libraries, support for Jena API during pattern matching, and the support for multiple stream processing. Other

types of RSP systems are either closed source or are not capable of supporting serialised RDF data. Also, based on the requirement for the semantic process to support data integration and semantic inference from the combination of heterogenous and related IoT streaming data, C-SPARQL suggests being the best candidate for the actualisation of the stream validation approach. The approach has used the syntax of C-SPARQL in figure 4.1 to achieve selection and ordering of semantic IoT streaming data available as serialised RDF streams at run-time through a continuous semantic query.

C-SPARQL has shown to be promising for the approach (SISDaV) as it presents an

```
FromStrClause  := 'From' [NAMED] 'STREAM' StreamIRI
                  '[ RANGE  Window ]'
Window         :=  LogicalWindow
LogicalWindow  :=  IntNumber TimeUnit WindowOverlap
TimeUnit       :=  'ms'|| 's'|| 'm'
WindowOverlap  :=  'STEP' IntNumber TimeUnit
ORDER          :=  'ASC' || 'DESC' [ ?variable ]
```

Figure 4.1: Syntax for IoT streaming Data Selection and Ordering

execution framework built on top of existing stream management systems and triple stores. The inclusion of the triple store provides an alternative for IoT stream framework to manage memory and disk utilisation during processing. This is because the triple store uses the dictionary encoding method to realise this task. SISDaV adopts the technique to reduce the semantic streams to make the stream fit into the memory during the semantic processing. It is however important to note that in the case of the IoT streaming data, data input done at a high rate may result in performance issues. Therefore, the remedy is to map RDF streams represented as binary digits to an identifier that is represented as a 64-bit integer. In principle, the first bit tells if RDF stream is encoded or not while the next five bits are used for the data type such as float or integer, etc. The remaining bits are used to represents that actual stream value. In cases where

the RDF stream cannot be represented as bits, it will have to be stored in the dictionary while the identifier is stored as the remaining sixty-three bits.

Furthermore, the framework uses a window-based strategy to access IoT streaming data with well defined IRI. In practice, the IRI represents the IP address and port that is used in gaining access to the streaming data. A window contains the latest sets of data streams from continuous streams that are to be considered in part by each query execution. The other reasons why C-SPARQL is considered suitable for the approach is due to its capability to perform nested aggregations and, supports for loading the bindings of the static part of the base SmartSUM ontology into the relations so that continuous queries over IoT streaming data are being executed against these relations. It also supports the closed world and time-aware reasoning with semantic rules over streaming data, without affecting any underlying domain knowledge or ontology. The Semantic Multiple Stream Processing addresses the requirements for Time consideration (**R1**), Data Integration (**R2**), Data-Driven Processing (**R3**) and Semantic Stream Querying (**R4**) .

4.3.3 Semantic Time-Aware Stream Validation Rules (*SenTAR*)

The use of data quality rules in defining methodologies for processing and detecting anomalies in data is an approach considered in ensuring high data quality (Li et al., 2011). Validity rules have been identified to govern quality issues related to data values, which specifically targets quality dimensions such as incompleteness, ambiguity, outdated and inconsistent data values (Adelman et al., 2005).

The validation rules used by SISDaV for the IoT streaming data quality validation borrows from the general concepts of interpolation and semantic integration. This facilitates the process to achieve semantic inference on streaming data produced by IoT nodes. These two approaches allow to performs stream validation for quality issues relating to IoT stream incompleteness, plausibility and inconsistencies including

structural differences in IoT streaming data. The rule specification for the semantic validation is based on heuristics and constructed from the established relationships among the set of variables or specific parameters present in the domain of application. This approach to rule specification provides inference based on the combination with corresponding values of external related data or events within the streaming space. For example, a validation rule can combine the streaming data from heterogeneous but related sensors including the status of actuators in a smart space to provide inference or validate IoT streaming data while also considering the time element of the streaming data.

Generally, rules derived from sets of assertions or axioms are usually based on established known facts or knowledge from domain experts defined from production logic such as Jena rule. Jena basic rule syntax is currently used for the implementation of both RDFS and OWL reasoners combined with individual native rules. These rules are developed for schema and static data within the knowledge graph model. A sensor-based rule for data processing in IoT applications can be developed using the Jena rule syntax. In the SISDaV a[[roach], the Jena rule language [1] is preferred as a declarative language to express the rules and evaluates matching against the semantic IoT streaming data. The reason is that the rule language supports the semantic web standards due to compatibility with OWL and RDF. Besides, its corresponding subsystems are based on Java, which is openly available thereby making it more adaptable to wide areas of applications. In the current SISDaV approach, instead of executing the rules against static ABox (schema for data individual within ontology model) as can be seen in traditional semantic web ontology reasoners, the continuous rules will be fired or evaluated against the run-time IoT streaming data.

It is a fact that IoT streaming data are usually boundless and cannot be stored before rules can be evaluated against them. An approach adopted by the SISDaV approach to address this challenge is by enhancing the Jena rule syntax to extract (using a binding

[1] Available: http://jena.apache.org/documentation/inference

on stream selection in section 4.3.2) a subset of the streaming data into a *time-based window* and then processing the data as an intermediate finite set of IoT streaming data by matching them against the validation rules. Due to the reason that the window is facilitated by the time component, it is necessary to include both the processing and time component as part of the rule language so that user can use them as part of the rules.

To enable the validation process to support the time-based window processing, the basic Jena rule syntax is extended with a new time and window components to accommodate the temporal characteristics of IoT streaming data. The original structure of Jena rule syntax mainly comprises of the Header, Rule and Node. The Header is used to locate and define the namespaces of the underlying RDF models and Ontology. The rule specifies the set of conditions including the primitives that are required by the ruleset. The Node indicates the actual variables, prefix and specific values that form the ruleset.

The extension of Jena syntax currently being introduced is shown in bold in figure 4.2. Ordinarily, Jena rules and syntax are applied to static data for reasoning purposes, in the current framework, it is adapted in a way to support reasoning over IoT streaming data. To allow the system to differentiate between the IoT streaming data and static data the STREAMWINDOW is introduced to define the subset of a finite set of IoT streaming data at per time. The STREAMWINDOW implementation of the rule is implicitly achieved with similar window processing infrastructure of RDF stream processing systems. Besides, Jena uses a native built-in primitive NOW(?X) to indicate the current system date/time values for rule instance but not for data stream instance. The new syntax addressed the issue by adding the TIMESTAMP to differentiate between the system time and the actual timestamps on each element of streaming data as they are contained within individual STREAMWINDOW.

```
Header         :=   URI...URI  ||
                    [ prefix   namespace ] ||
                    [ prefix   namespace : url ]

Rule           :=    bare-rule   ||
                     [ bare-rule ]  ||
                     [ ruleName : bare-rule ]

bare-rule      :=   term*streamWindow, ..., term*streamWindow
                  ----> hterm  ||    // forward strategy
                 bhterm   <----   term*streamWindow, ...,
term*streamWindow                        //backward strategy

term*streamWindow :=  (node*Window,  node*Window,  node*Window)  ||
                          //triple pattern
                     (node*Window,  node*Window,  functor)||
                          //extended triple pattern
                     builtin(node*Window,  ...  node*Window)
                              //invoke procedural primitive


Window  :=  dateTime (timeStamp ...timeStamp)
                       //selected window based on RSP

timeStamp  :=  variableTime, ...  variableTime
                               //value of annotated time

Functor  :=   functorName(node, ...  node)  //structured literal

Node     :=  uri-ref                      ||
             prefix : localname   ||   //e.g.  rdf:type
             ? variableTime   ||  // annotated timestamp on variable
             ? variable     ||    // vriable
             ' a literal '  ||    // a plain string literal
             ‘lex '^^typeURI||    // a typed literal
              number        // e.g. 45 or 54.8
```

Figure 4.2: Extended Jena rule Syntax (Extensions appears in bold)

Furthermore, the specification of STREAMWINDOW with the rule can be optional, this is because there are circumstances that rules with non-blocking operators may need to be processed without time component (e.g. filtering). The modified Jena rule syntax can be implemented or accessed by instantiating the GENERICRULEREASONER class. Specifically, the rule syntax is applied to develop domain-specific rules that will be used by the framework for validation of IoT streaming data for quality requirements relating to consistency, completeness and plausibility checks in an explicit manner (figure 4.3).

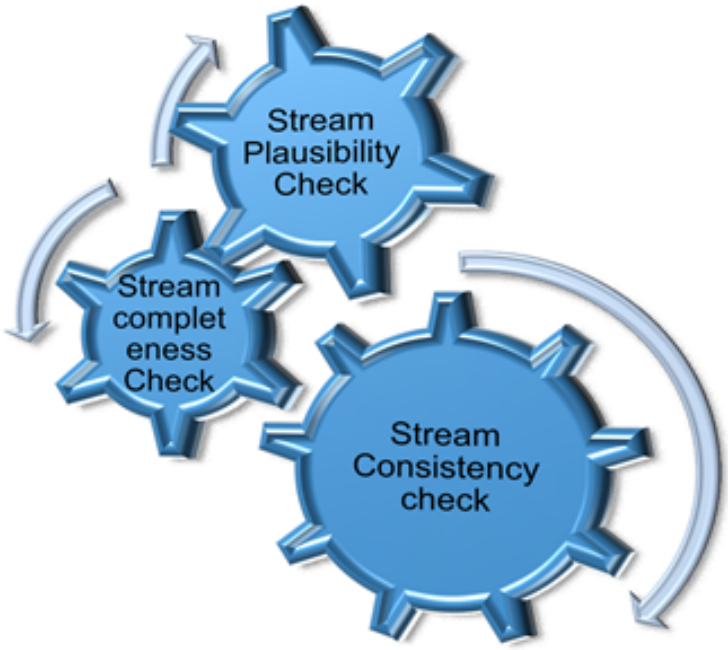

Figure 4.3: Sensor Streaming Data Validation Rules

4.3.3.1 Time Element of Jena Syntax

The time element specified in the extended Jena syntax is a set of primitive temporal entities for interpreting the time concept in semantic IoT streaming data. As previously introduced in section 4.3.1, temporal entities are *ordered* by ≤ relationship. For simplicity, the expression $t < t'$ and $t \neq t''$ to define a strict *order*.

Time element can typically be considered as either *Dense* or *Discrete*. A time element is *Dense* if is an infinite set and $\forall t, t' \in T$ with $t < t', \exists t'' \in T : t < t'' < t'$. On the other hand, time element will be considered as *Discrete* if it is a specific last element from a series and is assumed not to have an immediate successor with other elements having a predecessor except the first.

As an example, if an IoT application emphasises a fixed smallest granularity (e.g. seconds), the *Discrete* time element is the most appropriate choice. However, if a lower granularity such as rational number is required, then the choice of *Dense* time element is more suitable. The current assumption that the time elements in IoT applications follow some sort of linearity is much reasonable and most of these applications

will support time elements with rule-based processing approach.

4.3.3.2 IoT streaming Data Validation Rules

The following paragraph provides the generic rules required to validate the specific IoT streaming data quality problem as previously identified by this research. The rules are family of first-order predicate logic with forward inference strategy. The namespace adopted by the rule construct is defined by the URI of the SmartSUM model (section 3.3). The generic validation rule specified in this section specifically targets numeric IoT/sensor streaming values produced at the IoT node. Apart from the completeness rule that takes place at the entry point of the validation system, the other two rules are triggered at the point of the semantic reasoning process.

- **Completeness Rule** The completeness rule (also called Discrimination Rule) for semantic validation process is essential for ensuring missing data points within the streaming windows of IoT streaming data are spotted or possibly eliminated from a streaming window. Missing data points in IoT streaming data are often indicated by IoT applications as the replicated value of previous data point with the same timestamp (e.g. as seen in EEG instrumentation) within a streaming window. Other streaming IoT nodes can alternatively use a specific literal value or specific constant to indicate the missing data in a streaming window. To enable identify and validate each IoT streaming window for missing data, it is assumed that each streaming window is of fixed length based on the streaming rate. The procedure in Listing 5.1 embeds a discrimination rule to prune all identified incomplete streaming data that may likely be present during a streaming window. The Completeness/Discrimination rule consider the notion of discrete-time *t'* and triple patterns $\mathbb{T}^{pattern}$ for evaluating IoT streaming data. A streaming triple corresponds to (*s,p,o*) = $M \in (S_{Logical})$ with discrete timestamp (*t'*, *M*) with non-decreasing timestamps $t' \leq t''$ where *t'* is the current timestamp. A Discrimination rule is described by repeated matching of IoT streaming triples $\mathbb{T}^{pattern}$ of the form (*?a, p1, ?b1, t'*), (*?c, p2, ?b2, t'*), (*?x, p3, ?b3, t'*) representing IoT stream-

ing data, where textita, c and x are variables that represents subject of triples with the corresponding objects and timestamp. The matching is usually a done between the previous and current variables of an identical IoT streaming triple by using the subject-subject, object-Object and timestamp-timestamp mappings for the triple pattern. The STREAMWINDOW is only considered for the current streaming triples during matching. In the rule, $M, N \in S_{Logical}$ represents previous and current Streaming triples respectively with the individual timestamp. The matching technique of the rule is similar to the RETE matching algorithm (Forgy, 1989) except that it does not perform join operation during processing, as it only considers individual streaming triple elements for matching. It runs a $\alpha-memory$ which is temporary memory to holds intermediate streaming triples during matching operation. The discrimination rule executes against IoT streaming data represented as triples over stipulated streaming windows by matching the triple components of previous (M) streaming data against the current one (N) including the timestamps. The output of matching is propagated into the $\beta-memory$ or pruned from the STREAMWINDOW.

Structure : Discrimination Rule

LS is the streaming triple; M is previous streaming triple;

N is the current streaming triple; ω is the current window size,

$\mathbb{k} =' Fixedliteralvalue'$

t',t" are initial and current timestamps respectively

Procedure: *Read* (M, N)

1. *if* N = LS $\in \omega : \omega \in$ W

2. *load* N , M into $\alpha - memory$

3. *Split* M, N := $(s_M, p_M, o_M, t_M), (s_N, p_N, o_N, t_N)$ = ($\mathbb{T}^{pattern}$, t) $\in$ LS

Procedure: *Match-Prune* (N)

1. *while* (t' < t") $\wedge$ (N $\in \omega$)

2. *Match* $(s_M \simeq s_N; o_M \simeq o_N; t_M \simeq t_N)$

3. *if* {N:= M $\wedge$ N(t'):=M(t') } $\vee$ {N := $\emptyset$} $\vee$ { N:= $\mathbb{k}$}

4. *Remove* N from $\alpha - memory$

5. *else*

6. *set* $\beta - memory$:= N

7. nextN $\in$ LS

Listing 5.1: Completeness Rule

- **Consistency Rule**

  Based on the quality problems associated with sensor streaming data as discussed in section 2.2, inconsistency problem can arise when streaming data produced by sensor node applied to a specific context produces data that suggests being out of range in value, or there are disparities in the values of streaming data provided by similar IoT nodes deployed within the same space.

  The consistency rule showing in listing 5.2 is a forward rule and will execute during the reasoning windows. It consists of the rule body and the head. The rule body consists of a set of conditions denoted as C_i where i = 1, 2,3,...n. Each T_i is a rule set that is expressed as a triple pattern with a logical connector that labels the streaming values with its associated discrete timestamps produced within each streaming window. It consists of additional test statement that verifies the validity of the streaming values based on defined domain expert knowledge or specific user-defined values. There is also an expression containing a comparative operator that is used to check whether the timestamp of specific context value falls within the same timestamps of similar and related sensing nodes.

  The head provides the method and format for representing the output after inference has taken place by the reasoner. In other words, it represents the implication of the combination of the ruleset that was defined in the rule body following a forward inference strategy performed by the reasoner. The domain expert can craft or format the output in any way as desired. It is worthy to note the method part of the rule head (_action) can come from any of the available built-in primitives (*remove, drop or print*) of Jena rule syntax.

Structure :

C1,C2,C3···Cn are set of rule conditions.

X is the specific measurement value of the target triple with timestamp.

ϕ represents the nth related triple value with timestamp.

Ω represents set of logical operators that can exist among rule conditions.

Δ is a set of possible comparison operator for IoT measurement timestamp.

PROPruleMode := "forward"

1. [*consistencyCheck :*
2. C1 = { (?x namespace:p ?K) ∧(?x namespace:q ?xTimeStamp)
3. ∧ (lowerBound < x < upperBound) }Ω
4. C2 = { (?y namespace:p ?K) ∧(?y namespace:q ?yTimeStamp)
5. ∧ (lowerBound < y < upperBound) }Ω
6. C3 = { (?z namespace:p ?K) ∧(?z namespace:q ?zTimeStamp)
8. ∧ (lowerBound < z < upperBound) }Ω
9. ···
10. CN = { (?ϕ namespace:p ?K) ∧(?ϕ namespace:q ?ϕTimeStamp)
11. ∧ (lowerBound < ϕ < upperBound) }Ω
12. Δ(?xTimeStamp,yTimeStamp)Ω
13 Δ(?xTimeStamp,zTimeStamp)Ω
14. ···
15 Δ(?xTimeStamp,ϕTimeStamp)
16. ⇒ *_action*(?x namespace:p ' literal string' ‖ X)
17.]

Listing 5.2: Stream Consistency Rule

- **Plausibility Rule** Plausibility may arise from poor clarity in IoT streaming data, which are often due to some false positives present in the data. In a typical IoT environment, common sources of data are from different types of sensors which includes the interaction of *things*. As an example, the false positive or plausible value of sensor streaming data may result from the interference of temperature sensor deployed to measure the indoor temperature but was measuring the human body temperature close to the location of its deployment. In another dimension, a false positive reading can be perceived as measurements produced repeatedly by a battery-powered sensor that still produces streaming data despite faulty battery. To validate IoT streaming data against errors from plausibility, a forward strategy rule shown in listing 5.3 is required to interpolate and process external data related to the IoT streaming data of interest when the rule is triggered. The rule executes, based on outcomes of specific events that are identified to be peculiar to certain settings or domains. The outcomes or state of the events are usually processed as boolean types in the domain of applications. Since the values of sensor reading are interpreted as real or integer types aside that of actuators that interprets as boolean type, it makes sense for the domain experts to be able to set some threshold values at the time of deployment of the validation system. These values are represented as the *lowerbound* and *upperbound* variables in the rule syntax. The implication of the rule is similar to the consistency rule. It can assume any of the possible action defined as built-in primitives from the native Jena rule syntax such as *remove, drop or print*. The effect of this primitives is actually what the name implies.

Structure : C1,C2,C3···Cn are set of rule conditions.

X is the specific measurement value of the target triple with timestamp.

ϕ represents the nth related triple value with timestamp.

Ω represents set of logical operators that can exist among rule conditions.

Δ is a set of possible comparison operator for IoT measurement timestamp.

PROPruleMode := "forward"

1. *read()* Pre-Conditions:
2. *if* ((actuatorState == 'True')‖(motionDetected == 'True')‖
3. (sensorState == 'False')‖
4. (actuatorState == 'True' && motionDetected == 'True')‖
5. (actuatorState == 'True' && sensorState == 'False')‖
6. (motionDetected == 'True' ‖ sensorState == 'False')‖
7. (actuatorState == 'True' ‖ motionDetected == 'True')
8.) Then
9. [*plausibilityCheck :*
10. C1 = { (?x namespace:p ?K) ∧(?x namespace:q ?xTimeStamp)
11. ∧ (lowerBound < x < upperBound) }Ω
12. C2 = { (?y namespace:p ?K) ∧(?y namespace:q ?yTimeStamp)
13. ∧ (lowerBound < y < upperBound) }Ω
14. C3 = { (?z namespace:p ?K) ∧(?z namespace:q ?zTimeStamp)
15. ∧ (lowerBound < z < upperBound) }Ω
16. ···
17. CN = { (?ϕ namespace:p ?K) ∧(?ϕ namespace:q ?ϕTimeStamp)
18. ∧ (lowerBound < ϕ < upperBound) }Ω
19. Δ(?xTimeStamp,yTimeStamp)Ω
13 Δ(?xTimeStamp,zTimeStamp)Ω
20. ···
21 Δ(?xTimeStamp,ϕTimeStamp)
22. ⇒ *_action*(?x namespace:p ' literal string' ‖ X)]

Listing 5.3: Stream Plausibility Rule

The rules form the main building block for the semantic reasoning process of the semantic validation approach, and they are designed to execute explicitly in any smart space scenario where applications will have to rely on IoT streaming data for smart or intelligent processing at a higher level of context. The selection of appropriate rule for execution is also coordinated during the semantic reasoning activity.

The feature of the semantic time-aware validation rules satisfies the requirements for Time consideration **R1**, Data Integration **R2**, Inference Ability **R5** and Stream Quality Management **R6**.

4.3.4 Semantic Reasoning for IoT Streaming Data- ConTAR

Instead of developing a separate method for streaming processing from scratch, it is reasonable to consider enhancing a suitable existing semantic streaming processing system to deliver a semantic reasoning for IoT stream quality validation tasks. The method of layering a semantic stream processing system with some set of semantic rules is considered one of the possible ways of achieving effective semantic reasoning and inference. In order to realise a semantic reasoning system for IoT streaming data and to actualise a solution for dealing with quality issues in IoT streaming data, a centralised continuous stream reasoning system built on C-SPARQL and the continuous semantic stream validation rules (section 4.3.3.2) is developed. The new semantic reasoning system is known as a Continuous Time-aware Reasoning (*ContAR*) and is shown in figure 4.4.

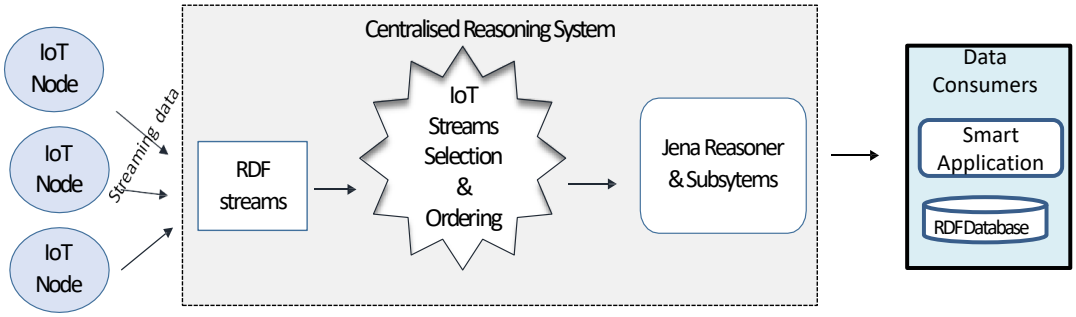

Figure 4.4: Centralised Reasoning Architecture

4.3.4.1 Continuous Reasoning System for IoT Streaming Data

After investigating the capability of various semantic rule-based reasoning systems to support IoT streaming data validation process, the GENERICRULEREASONER have been considered as most suitable for use. The reasoner is bounded to the IoT Streaming model available as a serialised data model. Apart from being able to reason over RDF schema, GENERICRULEREASONER supports the processing of the major RDF data serialisation format and serves as the underlying framework for C-SPARQL during schema matching, these features are missing in other semantic rule-based reasoners. As part of the development of the semantic validation approach, a Continuous Time-aware Reasoning system called *ContAR* have been considered as an essential aspect of the semantic validation approach. ConTAR combines C-SPARQL with the continuous validation rules developed in section 4.3.3.2 to achieve the continuous reasoning process while using the GENERICRULEREASONER for inference purposes. The GENERICRULEREASONER is a posteriori inference system that allows computing inference on-demand (i.e runtime) with the support of deductive rules and query processing. The reasoning system exploits the logical or sliding windows (Golab & Özsu, 2003) strategy to support the continuous reasoning process of semantic validation approach. The sliding window strategy allows semantic reasoning to be performed progressively at a given time interval that is usually shorter than the window's time interval. In that sense, ConTAR uses C-SPARQL for *Selection* and *Ordering* of the run-time IoT streaming data within each sliding window and allows the validation rules to execute against each window, based on the new time extension of the Jena rules.

4.3.4.2 Selection of Validation Rules for Reasoning Windows

The selection of specific validation rules will usually be triggered by certain conditions or events that are detected within an application context of the execution environment. Algorithm 1 recognises the three rules (in subsection 4.3.3.2) with the respective stream quality issues identified earlier in section 2.2. It relies on the specific state of event(s) that are occurring within the streaming space for the selection of specific rules

Algorithm 1: Semantic Validation Rule Selection

Input: eventState (E_{State}), deviceState (D_{State}), streamValue ($\mathbb{S}_{value}$) , $ID_{\mathbb{S}_{value}}$ timePoint (t) , rules ($r_1, r_2, r_3 \in \mathscr{R}$), $ID_{\mathbb{S}_{value}}$ and $Threshold(\Psi)$.

Output: Inferred Quadruple statement ($(s,p,o,t) \in Q$) with node $\alpha_Q \in (\mathscr{K}\mathscr{G})$ gets assigned to rule.

$(s,p,o,t) \in Q = r \in \mathscr{R}$;

repeat

  $\mathbb{S}_{Window} = \mathbb{S}_{value}, t, index$;

  $D_{State} = TRUE$;

  for *each* $(s,p,o,t) \in Q$ *within current streaming window* ($\mathbb{S}_{Window}$) **do**

    $r_1 = CompletenessCheck$;

    execute r_1;

    if $((E_{State} =' TRUE') \; AND \; (\mathbb{S}_{value} \neq \emptyset))$;

    then

      $r_2 = PlausibilityCheck$;

      execute r_2;

      break;

    end

  end

  if $((E_{State} =' FALSE') \; AND \; ((\mathbb{S}_{value} == q \in Q) \; OR \; (\mathbb{S}_{value} > \Psi)))$;

  then

    $r_3 = ConsistencyCheck$;

    execute r_3;

    break;

  else

    $r_2 = PlausibilityCheck$;

    execute r_2;

    break;

  end

  if $((E_{State} =' FALSE') \; AND \; ((\mathbb{S}_{value} == q \in Q)))$;

  then

    $r_1 = CompletenessCheck$;

    execute r_1;

    break;

  else

    $index$ ++;

  end

until $D_{State} =' False'$;

that will be triggered for the validation process. These type of events can vary from actions triggered by an actuator (e.g open/close window, doors, light, etc) to activities influenced by human beings such as standing close to temperature sensor meant for indoor temperature measurement. At the start of the semantic reasoning process, the algorithm executes the completeness rule on each quadruple statement that models

each IoT streaming data produced at run-time. The output is further matched against the conditions set for the plausibility check and executes r_2 if the condition is satisfied. r_3 is executed against the quadruple statement once the condition for r_2 is violated. The pattern continues until the status of the target streaming device is inactive or false. For multiple target devices or streaming data sources, the rule can be deployed in parallel over a distributed network nodes.

All instances of the semantic IoT streaming data that are subscribed by the reactive service with space, which does not violate the quality requirement will be later held by the KG through incremental data persistence. The semantic continuous reasoning feature of the framework allows Integration **R2**, Time consideration **R1**, inference Ability **R5** and Stream Quality Management **R6** requirements for semantic stream processing systems.

4.3.5 Incremental Data Persistence

This section provides an approach for management of inferred RDF statements as historical IoT data and also facilitates query over static IoT data. This feature of SISDaV approach operates outside the IoT streaming data validation environment discussed in the previous sections. One benefit of incremental persistent of the inferred triples with an annotated timestamp (called Quadruple statement) is to facilitate the provision of reliable and usable SPARQL endpoint that can support complete SPARQL query over archived or stored streamed data. Central to the process of persistence is a modification arising from continuous updates that relate to addition or removal of quadruple statements from the background KG, which the process must be able to support incrementally. This means the persistence of the inferred quadruple statements on the RDF graph (describing the KG) is done on an incremental basis as a new set of quadruples arrives on the graph. As a result, previous quadruple statements are joined with the recent statement using the *Subject-Subject* join.

In principle, statements produced from series of expired windows constitutes the ele-

ment of the incremental persistence on NoSQL storage. Suppose a set of $q_1, q_2 \in Q^e{}_1$ and $q_3, q_4 \in Q^e{}_2$ represents a successive inferred quadruple statements, and set ω_{e1}, ω_{e2} are representing corresponding expired windows respectively. At the initial process of persistence, the schema for the most recent statement $q_3, q_4 \in Q^e{}_2 \in \omega_{e2}$ is held within the temporary memory while the previously inferred statement $q_1, q_2 \in Q^e{}_1 \in \omega_{e1}$ is directly indexed and stored on the disk. During the subsequent persistence of $Q^e{}_2$, the set of quadruple statements in $Q^e{}_1$ is extended in the memory with statements from $Q^e{}_2$ using the *Subject-Subject* join operation which results in $q_1, q_2, q_3, q_4 \in Q^e{}_2$ and later indexed and stored on the disk. This process will continue to operate incrementally until the validation process is terminated. The storage of the new set of statements will override the previous quadruple statements to optimise storage and prevent overhead once a new session of semantic data validation begins. To provide a better view of the incremental persistence of the inferred quadruple, an architecture that adopts a native RDF data persistence (Blin et al., 2012) is introduced as shown in figure 4.5 .

Figure 4.5 contains multiple formats RDF data storage with partitioning for disk mapping to achieve the Incremental data persistence. During the process of data persistence, the RDF-based Web application with inference module provides an interface to initiate storage of expired IoT streaming data by using the RDF parser. RDF parser with the model registry can map the quadruple statement to a specific NoSQL database while also determining which NoSQL model translation the statement will adopt. The Dictionary informs the Partition Manager on how to determine the appropriate partition of the NoSQL database to store the inferred statement. The NoSQL database can be one of the columnar Database, Graph database, document database or Key/Value database. In this architecture, the columnar database is considered for implementation because it supports vertical partitioning that involves fast *Subject-Subject* joins, and column-oriented storage that benefits from compressibility and performance of not having to read entire row into memory from disk. In addition, the vertical partitioning approach to RDF statement storage offers benefits of multi-valued *Subject* attribute such as *tempReading* with multiple readings *Object* values in the RDF graph.

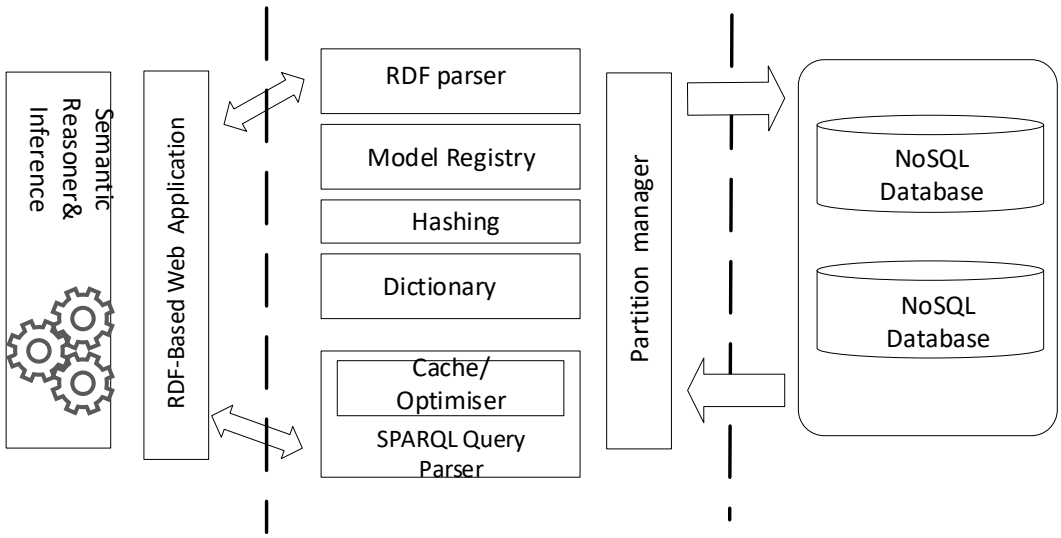

Figure 4.5: IoT Data Persistence Architecture

Similarly, query request initiated with SPARQL language is decomposed into sub-queries by Query Parser and Optimiser, which later report the outputs to Model Registry. It, in turn, looks for the component of the dictionary that contains the partition in which the query statement is possibly stored on the database. It uses the information to verify which statement are currently available in the Cache and interprets the SPARQL query for the missing statement to graph, columnar and/or document database queries. In the final part of the query processing, the SPARQL Query Parser uses the dictionary to convert the results back to the RDF statement for the RDF-Based web Application.
This approach further provides a foundation for the method of realising the incremental materialisation of the inferred triples. Incremental materialization (Urbani et al., 2013) suggests being a realistic approach to updating a knowledge graph with new fact especially in computationally expensive situations that involve dynamic and large data volumes such as IoT streaming data. Therefore the feature corresponds to the Historical Stream Management **R7** requirement for stream validation systems.

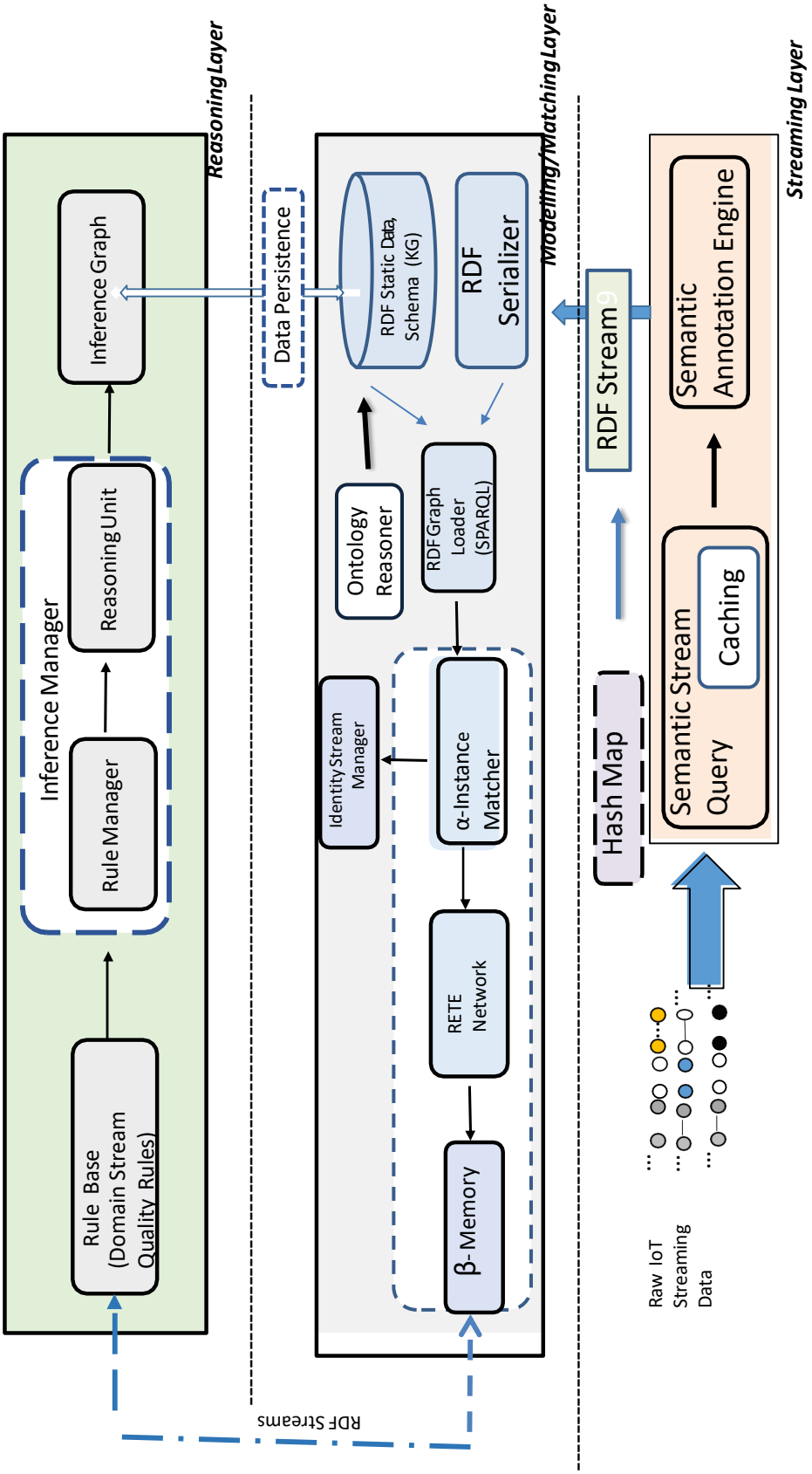

Figure 4.6: SISDaV Framework

4.4 Integration of *SISDaV* Approach as Unified Framework

The *SISDaV* framework is primarily based on two different technologies: the RDF stream processing system and the continuous stream reasoning system. These two technologies have been carefully outlined and integrated into SISDaV in a manner that allows run-time validation of IoT streaming data before it can be eventually consumed by reactive applications or other categories of Decision Support Systems. The architecture presents a unified semantic validation system which is available as API for software applications. In general, the SISDaV architecture is built on three layers as depicted in figure 4.6 by incorporating the features identified in section 4.3. All the layers are described in detail within the following subsections.

4.4.1 Streaming Layer

This layer forms a physical connection between the heterogeneous IoT streaming nodes such as sensors (Physical and Virtual sensors), actuators and other ubiquitous devices. The data produced by these nodes are usually of heterogeneous formats and contains certain temporal aspects. The layer typically consists of the Semantic Query and Semantic Annotation Engine. The raw IoT streaming data can be admitted into the architecture through the Java Message Service with the support of ActiveMQ Message broker. Further processing is performed by creating an index structure on each raw IoT streaming data using the hash table within the Hash Map module. This makes it possible to eliminate possible redundancies among raw streaming data due to overlapping streaming windows, by defining unique Id on the raw streaming values. Each streaming values are now converted into a unique RDF stream. The equivalent RDF stream is achieved using the data transformation with the support of Semantic annotation Engine during data pre-processing. The engine relies mainly on the vocabulary of the SmartSUM ontology to define the semantic streaming data. The semantic streaming data (also called Quadruple statement) are triple representation (subject, predicate object) with timestamps of the raw streaming data.

Semantic representation of the streaming data is necessary to facilitate the semantic query and multiple quadruple statement selection as required by the unified IoT streaming validation system. Selection or identification of semantic streaming data cannot be achieved by any of the native database query languages (such as SQL) or tools. It can only be realised with an equivalent dynamic semantic approach that can support the temporal characteristics of the stream. In this regards, the use of semantic continuous querying (C-SPARQL) is recommended for the stream selection. The continuous query is considered applicable for the semantic stream selection over continuous sliding windows during the semantic stream retrieval task.

To speed up the query processing, a caching method is introduced within the query processing module. This approach to querying semantic streaming data can support multiple and parallel selection of RDF representation of the IoT streaming data. The output of the semantic selection usually in form of semantic streams are pushed into the immediately upper layer of the architecture for further processing.

4.4.2 Matching Layer

The entry point to the matching layer is the RDF serializer which is responsible for converting the RDF stream into an alternative RDF data. The module works alongside the repository for the RDF schema containing the background knowledge graph with ontology-based reasoner used to maintain consistency from the new instance. The RDF schema is used by the RDF serializer to define the new RDF formats. The RDF serializer recognises the four types of serialisation format discussed in section 2.4.1. The importance of the RDF serializer module is to provide an encoding that will provide expressiveness and effectiveness with efficiency for semantic validation of the semantic IoT streaming data. The output of the module is supplied to the $\alpha-memory$ which forms part of the RETE network. One advantage of the RETE network is the ability to save state and computes partial results for future use without the need for re-processing

request when new semantic streams arrive at the module. In addition, it allows the network nodes to be shared thereby reducing the computing overhead within the memory. The $\alpha - memory$ executes the completeness rule (see section 4.3.3.2) to perform primary matching on each stream in order to check if the statement conforms to the pre-defined schema for all semantic streams. Any of the semantic stream that violates the matching process is classified as Identity stream and immediately removed from the current window by pushing it to the Identity Stream Manager that contains static methods for processing such stream. Otherwise, the output is received by the $\beta - Memory$ of the RETE network that forms interface with upper Reasoning layer and facilitates the secondary semantic matching process.

4.4.3 Reasoning Layer

The reasoning layer of the framework can access the serialised format of the semantic IoT streaming data from the matching layer through the $\beta - Memory$. The layer majorly consists of the Rule base, Inference Manager and the Inference Graph modules. The Rule base keeps knowledge about the current states of events related to actuators within the space, as well as the generic rules that are defined based on consistency and plausibility checks on IoT streaming data. The generic rules are modified or applied to suit specific events and IoT related domain where IoT streaming data are consumed by reactive applications. The selection of specific validation rule is determined by a special module known as the Rule manager which is within the Inference Manager.

Specifically, before semantic reasoning process, the rule manager relies on rule selection algorithm 1 to select specific rule that is suitable for the secondary matching and the eventual semantic reasoning and validation process. Usually the current states of all events are retrieved from the Rule base module and with the help of the RETE matching it can identify the specific statement for the plausibility check. Otherwise, the consistency rule will be executed against the semantic streams.

The reasoning engine unit is based on continuous reasoning approach described in section 4.3.4 by layering the window session of the stream query with a forward strategy rules stored in the rule base. It runs over a sliding windows by executing the selected rules over the semantic stream query window that contains the current snapshots of the semantic streams. This reasoning approach is facilitated with use of the open Jena library [2] and its corresponding subsystems. The reasoning process is further enhanced with BIND or SCHEMABIND calls on the sliding windows to ensure there is no information loss during the continuous reasoning process. The purpose of the reasoning is to produce new set of knowledge by a method of semantic inference while taking into consideration the time component of the data. The inferred knowledge represents the output of the validation process involving the raw IoT streaming data.

The validated output from reasoning unit is managed by the inference graph component of the architecture. The module holds an in-memory structure to temporarily store the intermediate RDF statement in form of a graph network before it is eventually stored permanently on the base Knowledge Graph. The approach to in-memory storage of the semantic statements adopts the method of Incremental persistence discussed in section 4.3.5, where the continuous inferred statements are added to the KG with Join operation before being stored on the RDF table and domain ontology. Typically, treating the IoT data streams as historical data during storage involves two separate approaches, the first one being through the method of incremental persistence on the RDF graph. The second storage operation involves the process of additive change for ontology evolution on the domain ontology (SmarSUM Ontology). The importance of this change is to enable the update of the SmartSUM ontology to reflect not only the new streamed IoT data but also new set of nodes that may have been added to streaming space. The ontology maintains consistency with Pellet reasoner, which is one of the native ontology-based reasoners.

[2] http://jena.sourceforge.net/

4.5 Conclusion

This chapter presented the main contribution of this work: the generic design requirements for IoT stream quality validation systems, and the unified architecture for the IoT streaming data quality validation approach, namely *SISDaV* as a rule-based semantic-driven approach with continuous reasoning. The design requirements are based on the common requirements of stream processing systems but with support for stream quality management for IoT/sensor Streaming data.

Furthermore, SISDaV provides semantic inference capability with the generic rules that natively extends the Jena rule grammar to enhance the continuous semantic reasoning operations. Adding semantics with RDF data to raw IoT streaming data and sensing nodes is expected to significantly improve the interoperabilty of heterogeneous IoT streaming data. Consequently, improve the quality validation of these streaming data to reduce the false positive alarms during actuation or decision making process. Also, the choice of RDF for modelling the sensor streaming data can leaves the programmer with the question of which serialisation format will be suitable for the sensor streaming validation. The alternatives RDF serializations identified in this section are developed to be used by web applications. However the aspect of its effects on IoT streaming data processing was not considered when these formats were designed.

The general stream processing systems including those for semantic applications has always been used to solve problems relating to data stream retrieval, high volume and latency of streaming data in the past. The approach implemented by this work extends the capability of existing semantic stream processing system with the feature that will support the quality validation of the IoT streaming data for reactive applications. In the next chapter, some concrete application cases studies are presented along with both effectiveness and efficiency evaluations of the framework.

Fifth Chapter

Evaluation of Approach and the Unified Framework

5.1 Introduction

This chapter provides a detailed comparative evaluation of the proposed semantic IoT Streaming data Validation approach and its framework presented in the previous chapter. This is required to enable satisfy the fifth research objective and, provide answer to the research questions **RQ4** and **RQ5**. Evaluation Metrics for effectiveness and efficiency of the semantic validation approach for the case study in smart home and smart city applications are provided in section 5.2. Section 5.3 describes the Smart Home Automation case study including the related sensor streams data set(section 5.3.1). This is followed by the description of the prototype architecture implementation in section 5.4. The second application case study evaluation is contextualised in the smart city project with air quality dataset in section5.7 The remaining part of the chapter provides the detail discussion of results and conclusion of the chapter in sections 5.8 and 5.9 respectively.

5.2 Evaluation Metrics

The metrics described in this section provides a means to evaluate the effectiveness and efficiency of *SISDaV* through its integration and deployment in a relevant use case

scenarios. These metrics are derived from the well established precision, recall and accuracy metrics (Powers, 2020), which has recently becomes the means of evaluating framework for IoT streaming data (Balakrishna et al., 2020). Furthermore, the evaluation metrics will be applied to the domain of Smart Home IoT-based control system (Wang et al., 2013) and Smart city Decision Support System (D'Aniello et al., 2018) later in this thesis.

5.2.1 Effectiveness Evaluation Metrics

The effectiveness of the framework is determined by the estimation of *Relevance score* and *Validation Accuracy* of the intervals of validation cycles involving the serialised RDF formats. The *Relevance score* provides an estimate of how precise the framework can identify the actual inconsistent streaming data with the occurrence of plausible data in each cycle. In the same manner, the *Validation Accuracy* is an imbalanced accuracy that estimates the combined effects of incomplete streaming data (True Negatives) and Plausible streaming data on the validation of inconsistent streaming data. The evaluation will further conducts a comparison of the four alternatives of RDF data serialised formats based on the results of the estimations.

The method of deriving the effectiveness of the framework is similar to what is currently used Information Retrieval (IR). This similarity is seen in terms of classification problem, which also adopt a similar technique for pattern matching during the semantic query and reasoning process. These methods of estimation express and compute the ratio of relevant and inconsistent instances of sensor streaming data among raw IoT stream instances within each streaming window. However, the notion of recall is estimated as an expression of the accuracy of the validation score. More importantly, since the cost of false negatives (in this case, represented as consistent streaming data) is considered to be very low in this context as it is expected that an IoT streaming node will produce more consistent values for the majority of the streaming lifetime, which results in a limited search space.

The metric considers the influence of plausible streaming data for each validation cycles of Inconsistent streaming data. This provides a practical method of estimating the exact or precise effectiveness of validation process in an IoT streaming environment where these type of stream quality problems persists. A similar example is a single smart home sensor such as temperature sensor that produces false-positive data and outliers at different streaming windows, which can ultimately result in a false alarm. In such a case, the estimation technique must be able to cater to both types of erroneous data when determining the number of relevant streaming windows and the accuracy of the system. Adapting the statistical method of estimation expressed as the ratio of true positives to overall false positives with true positives, We define the *Relevance Ratio* as follows:

$$\textit{Relevance Ratio} = \frac{\sum_{i=1}^{N} IC_i}{\sum_{i=1}^{n} IC_i + PC_i} \tag{5.1}$$

where, IC represents the inconsistent streaming data per streaming window S_W, PC represents plausible value (i.e total false positive values) and, both N and n are the total Inconsistent count and combined overall plausibility with Inconsistent count for all validation windows respectively.

Similarly, to obtain the estimate of the validation Accuracy of the framework over every successive validation cycle. It becomes necessary to consider the effectiveness of the validation process for inconsistent streaming data in the presence of plausible or incomplete streaming data, because of the cost and possible multi-class Class Classification problem associated with data during validation cycles. The estimation or evaluation metric recognises the imbalance or Asymmetric nature in the distribution of the streaming data per cycle. Therefore, it first computes the Sensitivity (TP_R) of each

validation cycle to determine the true positives rate as follows;

$$TP_R = \frac{\sum_{i=1}^{n} IC_i}{\sum_{i=1}^{n} IC_i + CR_i} \tag{5.2}$$

where, *IC* and *CR* are respectively the number of Inconsistent and Consistent streaming window in every cycle.

Secondly, an estimation of the Specificity (TN_R) rate is derived to understand the total Specificity (TN_R) per validation cycle. This estimation represents the number of true negatives per validation cycle and is derived as follows;

$$TN_R = \frac{\sum_{i=1}^{n} NC_i}{\sum_{i=1}^{n} PC_i + NC_i} \tag{5.3}$$

where *NC* and *PC* corresponds to the number of Incomplete/missing and Plausible streaming data produced within each cycle respectively. By combining both equations 5.2 and 5.3, the resulting validation score as a measure of the balanced accuracy for each validation cycle is therefore given as;

$$Validation\ Accuracy = \frac{(TP_R + TN_R)}{2} \tag{5.4}$$

The validation score estimation will be able to normalise the distribution of the validation output produced from the imbalanced or Asymmetric streaming data set. In addition, it will facilitate the comparison between actual raw streaming data and annotated validated outputs by the framework.

5.2.2 Efficiency Evaluation Metrics

Efficiency Metrics further evaluate the performance of the framework as a means of its cost on the system resources in which it is deployed. In an IoT domain that is heavily dependent on streaming data for data-driven processing such as reactive services, the importance of time is inevitable. As such, systems or applications will require to operate promptly with possible limited resources. For this reason, the efficiency metrics

considered for the evaluation of the application case studies are based on time. The time-based metrics include the Semantic reasoning time, Processing time and, Latency of the semantic validation approach with the framework.

Semantic Reasoning time estimates the time interval between the receipt of the serialised RDF data and when inference is produced. The semantic reasoning time is exclusive of the time it requires to perform ontology reasoning on the embedded domain ontology during ontology evolution as this is performed separately by the native ontology reasoner. It specifically measures the total duration to perform inference on each semantic streaming data per streaming window within every validation cycle.

In a similar perspective, the processing time is a combined estimate of the total duration that the semantic framework will require to process or validate the raw IoT Streaming data. This includes the time requires to perform a semantic query for semantic stream selection and, the time it requires to complete the semantic reasoning task.
The processing time is computed for individual streaming data per streaming window within each validation cycle. In principle, suppose S_d represents an individual semantic streaming data within a validation Cycle (V^{cycle}), R_{Time} and Q_{Time} represents reasoning time and semantic query time respectively. Therefore, $\forall S_d \in V^{cycle}$

$$ProcessingTime = R_{Time} + Q_{Time}$$

Latency is used as a performance metric to estimate the total delay between intervals of response time during the validation process. It is the total time taken by the framework to produce the first request from streaming data validation. In order words, it estimates the duration of delay experienced by the framework between intervals of semantic streaming data validation cycles. The metric describes the trend in the performance over intervals of validation cycles.

The metrics for effectiveness and efficiency evaluation described in this section will be applied for the evaluation of the use case scenarios described later in this chapter. The

interpretation of the performance testing will be interpolated over validation cycles for each of the RDF serialised formats considered in the experiments.

5.3 Case Study 1: Smart Home Automation System

The present application case study considers the case of a smart home IoT-based control system (Wang et al., 2013) with a corresponding dataset in smart home[1]. A number of issues associated with the characteristics and outputs from sensors and actuators are found to be relevant to the features of the smart home automation initiative. The motivation for the choice of solution to the case study is necessitated by the need for effective and efficient management of the appliances and operations that are mostly coordinated through the data managed by SHIS. Focusing on this component is a way of maintaining continuous interoperability with clean data processing requirements for the smart controller and actuator within the smart home. Web-based data modelling and processing for interoperability of various data produced within the smart home environment can only be guarantee through the semantic approach. This can play major aspects in the actualisation of a fully functioning and accurate smart controlling system.

Again, plausible values represent sensor readings that are greater than the initial controller set point or permissible range of values. Plausible readings in the case study are values which may have been compromised by external influence e.g. in the case study, a sensor reporting the value of body temperature/outdoor temperature instead of indoor air temperature. The objective of the Home Automation scenario is to evaluate the effectiveness and efficiency of SISDaV at different conditions of Inconsistent data points per streaming window.

[1] https://www.refitsmarthomes.org/datasets/

5.3.1 Description of Smart Home Dataset

The dataset used for the evaluation of the Home automation use case is currently obtained from the REFIT Smart Home project. One of the goals of the data set is to inform of the Internal environmental conditions and energy demands in UK homes. The data was collected from twenty homes with Smart Home technologies that include a series of sensor measurements. A brief description of the models and type of sensor technology are presented in table 5.1.

As reported by the contributors of the project, some of the data from the air temperature sensors contain duplicated and missing data points. Data were gathered from a total of 1,567 sensors that produced a total of 25,312,397 raw sensor readings with timestamps. The description of the smart home data providing the measurements for property of context such as temperature, humidity, pressure, door and window status are provided as follows

- Sensor_ID: unique identity of each physical sensor deployed within the home
- Start_time: time instant that mark the beginning of producing the sensor reading for each stream window.
- End_Time: elapse time instant before the start of the next sensor reading
- Manufacturer: brand name of the sensor manufacturer
- Measurement_ID: unique Id of each sensor reading or measurement value
- DateTime_value: date and timestamp associated with each measurement
- Temperature_value: actual value of air temperature reading
- Humidity_value: actual value of the indoor humidity
- Pressure_value: value of the indoor pressure within the smart home
- Window_ status: current state (open/close) of the window as controlled by an actuator

Table 5.1: Description of sensor types with measurement values

Sensor_ID	Start_Time	End_Time	Manufacturer	Model	Measurement_ID
Sensor41	2013-10-02T05:00:00Z	2013-12-03T15:15:00Z	Onset	Hobo pendant	TimeSeriesVariable41
Sensor42	2013-12-03T18:00:00Z	2014-06-18T10:45:00Z	Onset	Hobo pendant	TimeSeriesVariable42
Sensor43	2014-06-18T11:00:00Z	2015-02-27T14:45:00Z	Onset	Hobo pendant	TimeSeriesVariable43
Sensor44	2015-02-27T15:00:00Z	2015-08-14T09:15:00Z	Onset	Hobo pendant	TimeSeriesVariable44
Sensor29	2013-10-02T05:00:00Z	2013-12-03T15:15:00Z	Onset	Hobo pendant	TimeSeriesVariable29
Sensor30	2013-12-03T18:00:00Z	2014-06-18T10:45:00Z	Onset	Hobo pendant	TimeSeriesVariable30
Sensor31	2014-06-18T11:00:00Z	2015-03-02T09:45:00Z	Onset	Hobo pendant	TimeSeriesVariable31
Sensor32	2015-03-02T10:00:00Z	2015-04-07T13:15:00Z	Onset	Hobo pendant	TimeSeriesVariable32
Sensor1294	2014-07-23T15:09:00Z	2015-04-28T13:13:00Z	RWE	Interior motion detector	TimeSeriesVariable2111
Sensor1294	2014-07-23T15:09:00Z	2015-04-28T13:13:00Z	RWE	Interior motion detector	TimeSeriesVariable2193
Sensor21	2013-10-02T05:00:00Z	2013-12-03T15:15:00Z	Onset	Hobo pendant	TimeSeriesVariable21
Sensor22	2013-12-03T18:00:00Z	2014-06-18T10:45:00Z	Onset	Hobo pendant	TimeSeriesVariable22
Sensor23	2014-06-18T11:00:00Z	2015-02-27T14:45:00Z	Onset	Hobo pendant	TimeSeriesVariable23
Sensor24	2015-02-27T15:00:00Z	2015-08-14T09:15:00Z	Onset	Hobo pendant	TimeSeriesVariable24
Sensor33	2013-10-02T05:00:00Z	2013-12-03T15:30:00Z	Onset	Hobo U12	TimeSeriesVariable33
Sensor33	2013-10-02T05:00:00Z	2013-12-03T15:30:00Z	Onset	Hobo U12	TimeSeriesVariable432
Sensor33	2013-10-02T05:00:00Z	2013-12-03T15:30:00Z	Onset	Hobo U12	TimeSeriesVariable576
Sensor34	2013-12-03T18:00:00Z	2014-06-18T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable34
Sensor34	2013-12-03T18:00:00Z	2014-06-18T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable433
Sensor34	2013-12-03T18:00:00Z	2014-06-18T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable577
Sensor35	2014-06-18T10:00:00Z	2015-03-02T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable35
Sensor35	2014-06-18T10:00:00Z	2015-03-02T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable434
Sensor35	2014-06-18T10:00:00Z	2015-03-02T09:30:00Z	Onset	Hobo U12	TimeSeriesVariable578
Sensor36	2015-03-02T10:00:00Z	2015-08-14T09:15:00Z	Onset	Hobo U12	TimeSeriesVariable36
Sensor36	2015-03-02T10:00:00Z	2015-08-14T09:15:00Z	Onset	Hobo U12	TimeSeriesVariable579

Table 5.2: Extract of Raw Indoor Air Temperature Values

Measurement_ID	DateTime_Value	Temperature value
TimeSeriesVariable41	2013-10-02T05:00:00Z	17.772
TimeSeriesVariable41	2013-10-02T05:30:00Z	18.081
TimeSeriesVariable41	2013-10-02T06:00:00Z	18.176
TimeSeriesVariable42	2013-10-02T06:30:00Z	18.176
TimeSeriesVariabl41	2013-10-02T07:00:00Z	18.105
TimeSeriesVariable43	2013-10-02T07:30:00Z	18.01
TimeSeriesVariabl30	2013-10-02T08:00:00Z	17.891
TimeSeriesVariable39	2013-10-02T08:30:00Z	17.772
TimeSeriesVariable40	2013-10-02T09:00:00Z	17.701
TimeSeriesVariable40	2013-10-02T09:30:00Z	17.677
TimeSeriesVariable40	2013-10-02T10:00:00Z	17.677
TimeSeriesVariable22	2013-10-02T10:30:00Z	17.701
TimeSeriesVariable29	2013-10-02T11:00:00Z	17.701
TimeSeriesVariable41	2013-10-02T11:30:00Z	17.843
TimeSeriesVariable43	2013-10-02T12:00:00Z	17.938
TimeSeriesVariable43	2013-10-02T12:30:00Z	17.986
TimeSeriesVariable32	2013-10-02T13:00:00Z	18.01
TimeSeriesVariable40	2013-10-02T13:30:00Z	18.057
TimeSeriesVariable39	2013-10-02T14:00:00Z	18.081
TimeSeriesVariable32	2013-10-02T14:30:00Z	18.247

The dataset also contains climate data that were collected at the Loughborough University campus weather station. These data describes measurements relating to atmospheric temperature, pressure and relative humidity.

Specifically, the case study considered only aspects of data describing measurements from the room air temperature, relative humidity and pressure, interior motion detectors, door and window opening sensors and smoke alarms. The description of the relevant data set for the experiments including the description of sensor types, sensor manufacturer, sensor models and measurement Ids are presented in table 5.2. A snapshot of measurement values of the indoor air temperature with individual timestamp and measurement id considered for the case study are presented in table

5.4 Prototype Framework Implementation

Considering the importance of the sensor streaming data in the case study (presented in section 5.3), a prototype software architecture is developed as presented in figure 5.1. It is used to validate the sensor streaming data for optimal processing and controlling activities by smart home actuators. The design of the prototype architecture is based on *SISDaV*, which has been described in section 4.4. It consists of four layers with each upper layer relying on the immediate lower layer for input. The description of each layer and the interaction with the upper layer during the validation process are presented in the following sub-sections.

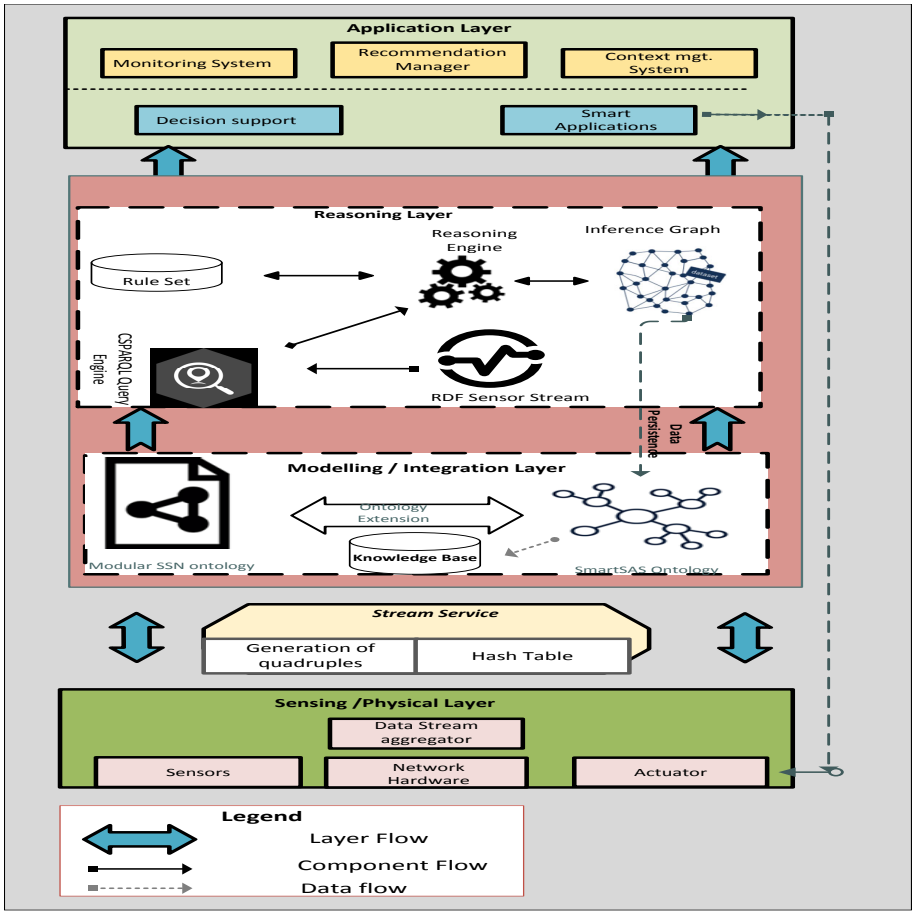

Figure 5.1: Prototype Architecture

5.4.1 Sensing/Physical Layer

The layer acts as an interface between the physical devices including sensors and the streaming data. It contributes to the large volume of data stream produced within the Smart Space. It consists of sensors and physical network devices. The Data Aggregator receives streaming data from different sensor nodes within the space with the use of MQTT protocol[2]. Data Aggregator pre-processes the data for semantic validation with the support of the sub-component integrated with this layer. Flexible data aggregation and delivery between physical layer and the immediate upper layer of the architecture are managed by Apache camel[3]. Apache Camel is a lightweight data/file transfer framework that allows integration between different components of a system. It accepts streaming data in any serialised format and routes the data between modules of the architecture. Most of the data produced at this layer come with temporal characteristics and are heterogeneous based on their representation by physical sensors producing the values.

In an attempt to provide a suitable data model for semantic querying and reasoning, each of the sensor streaming data is annotated with the support of RDF manager using the SmartSUM ontology that describes the smart spaces domain. The annotation converts each data to triple statement with the timestamps (Quadruple statement) by extracting the namespace including class and property types from the SmartSUM ontology. The resulting statement is read as RDF stream, which is further processed by the Stream Service. The stream Service runs in the background to re-write the RDF data using the native Jena RIOT API[4] to convert the RDF streams into alternative RDF serialization formats to achieve expressivity and faster processing. It also resolves the possibility of redundant quadruples statement by using the hash table indexing key/-value to provide unique identification for each quadruple. This is later facilitated for further processing by the upper layer of the architecture.

[2]Available: http://mqtt.org/
[3]Available: http://camel.apache.org/
[4]https://jena.apache.org/documentation/io/rdf-output.html

5.4.2 Modelling and Integration Layer

The main function of this layer is to provide interoperability among various heterogeneous IoT nodes and facilitates continuous integration of the sensor streaming data. The layer consists of the domain ontology model and its equivalent RDF graph for data persistence. In the scenario, the ontology is used to model the IoT nodes, physical properties and its observation data within the smart home. The observation data of interest include the temperature, pressure, humidity readings and actuators with related smart home objects. The domain ontology can support the addition of a new sensor node through its evolving functionality while maintaining the consistency among concepts through the pellet reasoner API. The ontology is also able to model the data from the new sensor node through relationship construction of data properties and concepts on the ontology model.

Another important use of the domain ontology is to provide the background knowledge necessary to facilitate continuous semantic reasoning and inference generation process. During the run-time sensor streams generation, the layer relies on the domain ontology and system memory to ensure the generation of RDF streams. It will eventually be applied to facilitate the run-time selection of semantic sensor streaming data in the upper reasoning layer.

5.4.3 Reasoning Layer

The entry point to this layer is the Query Engine. The engine is integrated with C-SPARQL query language for the continuous selection of multiple streaming quadruple statement over a continuous streaming window. It adopts window-based processing to support run-time RDF streams selection. This is performed by executing the listing in 5.2 for the smart home case study. During the C-SPARQL query processing stage, continuous pattern matching of concepts and properties are performed on each semantic statement describing physical properties, which includes temperature, pressure and humidity and the corresponding values and timestamps. The selection of the streaming

data variable within each streaming window is ordered by timestamps and executed over a period of 25 seconds with step interval of 7 seconds for indoor temperature values and 7 seconds for other related physical properties (indoor pressure and humidity values). The execution interval of the query is kept reasonably short to allow the system

```
REGISTER QUERY sensorValueOf  AS
PREFIX smartSpace: http://localhost:8080/smartSpace#
?pressureValue ?humidityReadings ?humidityValue "
  SELECT *
  FROM STREAM http://localhost:8080/smartSpace/streamTemperature [RANGE 25s STEP 7s]
  FROM STREAM http://localhost:8080/smartSpace/streamPressure [RANGE 25s STEP 7s]
  FROM STREAM http://localhost:8080/smartSpace/streamHumididty [RANGE 25s STEP 7s]
       WHERE {
        ?tempReadings smartSpace:hasValue ?tempValue.
        ?tempReadings smartSpace:hasTimestamp ?tempTime.
        ?tempReadings smartSpace:hasId ?tempId.
        ?tempReadings smartSpace:hasSeason ?tempSeason.
        ?tempReadings smartSpace:hasTimestamp ?tempTime.
        ?pressureReadings smartSpace:hasPressureReading ?pressureValue.
        ?pressureReadings smartSpace:pressureHasTimestamp ?pressureTime.
        ?humidityReadings smartSpace:hasHumidityReading ?humidityValue.
        ?humidityReadings smartSpace:humidityTimestamp ?humidityTime.
       }
      ORDER BY ASC(?tempTime)
```

Figure 5.2: C-SPARQL Query for selection of Sensor Streaming Data

to be subjected to stress test in a way to investigate its support for categories of sensors (such as location/movement sensor) with high streaming rate. Also, this will enhance the increase in detection rate within the sliding windows. It is expected that using a shorter step length than the streaming window length will result in some unnecessary duplication of the query result. The results from each query processing window are received by the ActiveMQ broker which is later managed by Java Message Service (JMS) and subsequently processed concurrently by the reasoning engine.

The reasoning engine implements an in-memory structure of Jena API used to temporarily hold semantic sensor streams from the current window during reasoning operation. The reasoning engine can access the serialised format of the semantic IoT streaming data from the current window with the support of RDF API to enable the continuous processing of the chosen serialisation format. The reasoning engine works in conjunc-

tion with the rules from the ruleset, which are defined based on domain regulations. Specifically, in the smart home case study base its validation rules on domain knowledge from occupational health and safety[5] recommendation for indoor temperature, pressure and humidity.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#
[consistencyCheck:
        (?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
        (?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
        greaterThan(?humidityValue,39)
        lessThan(?humidityValue,51)
        (?tempReadings smartSpace:tempHasTimestamp ?tempTime)
        (?tempReadings smartSpace:hasValue ?tempValue)
        greaterThan(?tempValue,17)
        lessThan(?tempValue,24)
        (?pressureReadings smartSpace:hasPressureReading ?pressureValue)
        (?pressureReadings smartSpace:pressureHasTimestamp ?pressureTime)
        greaterThan(?pressureValue,750.1)
        lessThan(?pressureValue,761.0)
        le(?tempTime,?humidityTime)
        le(?tempTime,?pressureTime)
        ->
        (?tempReadings smartSpace:isValid 'Consistency Check')
]
```

Figure 5.3: Consistency Validation Rule for Temperature Stream

The sample validation rule in figure 5.3 is used for checking the consistency in temperature readings. Specific validation rules for possible plausible IoT streaming data resulting from outdoor temperature and indoor event interference are listed in appendix D. The semantic reasoning module also implements the rule selection algorithm to determine the appropriate rule to execute from the ruleset based on an event that is predominant within the space during sensor streaming. It is configured to process snapshots of the semantic streaming data based on the domain background knowledge combined with the chosen rule using the forward inference strategy. The reasoning engine executes at run-time to provide sensor streaming values that satisfy the conditions

[5]http://www.ohsrep.org.au/hazards/workplace-conditions/heat

in the rules and make such available to web applications (in this case IoT-based Control system interface) through API. Finally, the output of the reasoning process is a new knowledge available as a quadruple statement. The quadruple statement or inferred RDF quadruple is added to the existing RDF graph using the method of Incremental persistence. The new inferred knowledge and streaming value can also be retrieved through the SPARQL query as static data.

The prototype implementation is realised based on three different java libraries. Specifically, the libraries include Jena[6], C-SPARQL[7] and JSON[8]. Each of the libraries is implemented as a plug-in and integrated with a cloud-based Microsoft Azure platform to demonstrate the feasibility of the semantic IoT streaming data validation approach. The platform allows for continuous integration and validation of sensor streaming data at run time.

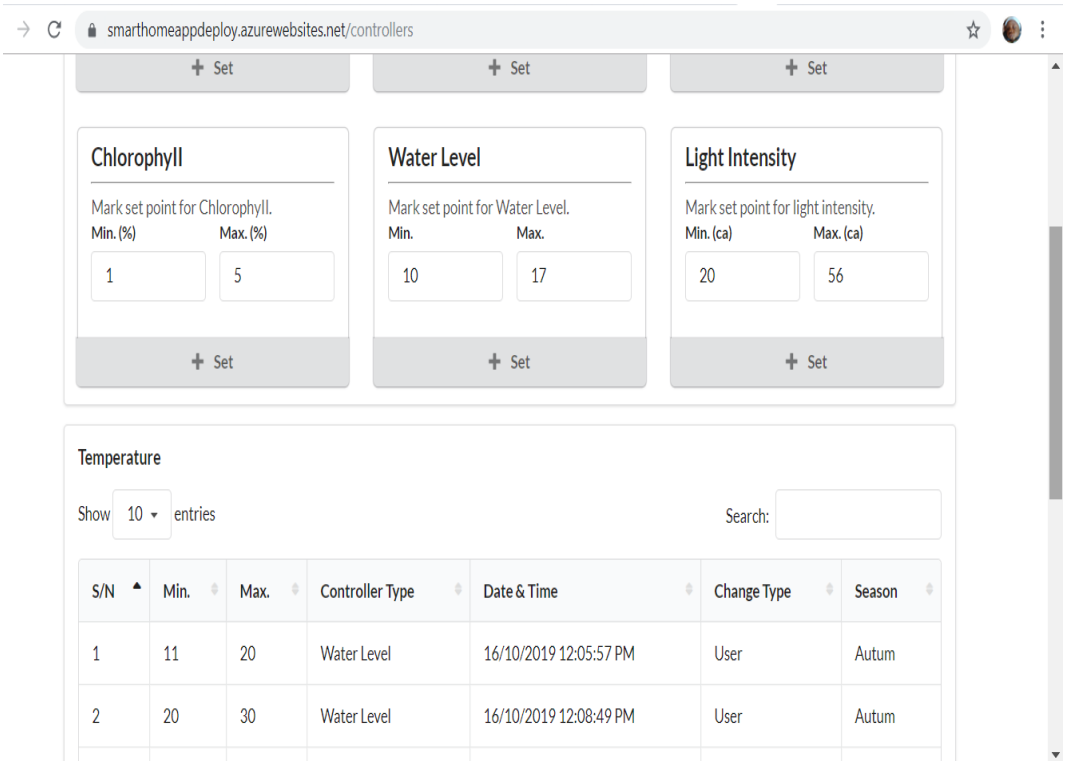

Figure 5.4: Section of run-time Temperature Values

[6]https://jena.apache.org/download/apache-jena-3.15.0
[7]http://streamreasoning.org/resources/c-sparql/CSPARQL-ReadyToGoPack-0.9.6
[8]json-simple-1.1.1.jar

To deploy the architecture in a way that is suitable for IoT-based controlling system in the home automation case study, a dynamic endpoint for user computer and the cloud-based server has been created. This makes it possible for the server to process data from a remote machine. The endpoint also facilitates the user's request through a personal computing device to remotely create a set point for the smart control system or get the status of each sensor within the smart home environment. The figures 5.4 and 5.5 contains the implementation of the user interface that keep track of the controller set points and the real-time analytics of the validation process respectively. More detail screenshots of the implementation are contained in Appendix B.

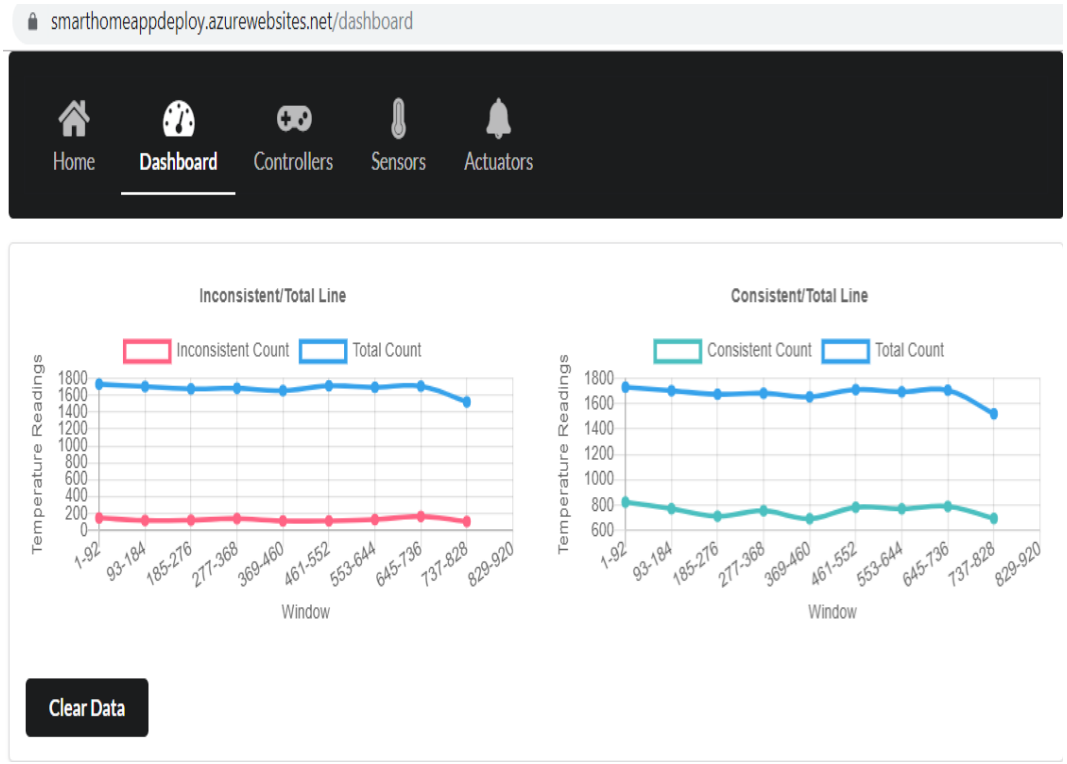

Figure 5.5: Real-Time Data Validation Analytic

5.5 Experiments

The aim of the experiment with the smart home scenario is to understand the effects of error rate and streaming rate of raw IoT data, on the effectiveness and efficiency of the semantic data validation approach. To allow for proper evaluation of the approach including its framework with the smart IoT-based Control system in home automation,

consideration is given to the run-time quality validation of the sensor streaming data. In the experiment, the focus is placed on the range of values published by smart home data set (refer to section 5.3.1). In particular, values produced by air temperature sensors, relative humidity and pressure sensor, door and window sensors, motion sensor and climate data. This data are reproduced in an experiment through a simulation library to allow for detail analysis of the framework in a real-life and run-time manner and applications. The detailed analysis data were collected in real-time for evaluation during each stage of the experiments. Given the importance of time and the influence of climatic season on the sensor streams, the experiment and evaluation approach considers both notions during the implementation and experimental process.

5.5.1 Sensor Streaming Data Generation

In a smart space environment (e.g. smart home), access to streaming data is usually through the deployment and configuration of physical sensors and related IoT nodes. In the current study, this practical approach seems to be a more expensive, time-consuming and labour-intensive task, which cannot be accommodated into the research. Therefore, to help determine the effectiveness and efficiency of the approach with the framework, real-time sensor streaming data are simulated using the C-SPARQL streamer library. The library is used to generate sensor streaming data with the individual associated timestamps using the current computer system time and date. In this way, it was possible to have access to the run-time streaming data and perform the run-time streaming data validation process. The generated values represent the streaming data produced by several different sensors present within the smart home based on the openly published data described by the open REFIT smart home data. In particular, outputs from ten sensor nodes that measures temperature, pressure, humidity, door open/close, window open/close, the motion of body and climate data are simulated. Specifically, the configuration of the sensors consists of 3 temperature sensors, 2 humidity sensors, 2 pressure sensors, 1 door sensor, 1 window sensor and 1 motion sensor. Correspondingly, each class of sensors generates the related streaming

data for the property it measures at a different streaming rate measured in seconds.

5.5.2 Experimental Setup

The experimental set up was conducted on a single node centralized server running on multiple processor computer (Pentium Core (TM) i7-4770 CPU @ 3.40GHz – 16GB RAM). The background memory structures that were allocated to the processes consists of an initial and maximum heap size memory of 1024m and 2048m respectively. The experiment involves a simulation of sensor streaming session that consists of eight rounds in total and with a duration of six hours for each round of the experiment. The total round of experiments was completed in two separate experimental runs. Each of the two experimental runs is allowed to perform semantic validation on the same type data set generated by streamer using the four RDF serialised formats.

To enable a better view of the performance of the framework in terms of its ability to continuously perform the validation process over the sensor streaming data, all the categories of sensor configuration in section 5.5.1 are employed. Each of the temperature sensor nodes is simulated to generate both true values and erroneous streaming values that explicitly represents typical Inconsistent, Plausible and missing readings respectively. Inconsistent streaming data values were injected into specific streaming windows at different intervals of a single (1) and ten (10) data points in separate experimental runs

Raw streaming data produced in both experimental runs represent outputs from physical sensor nodes and are implemented from separate concurrent java threads. Error injection rates of inconsistent streaming data were also completed in two separate experimental runs. Specifically, the evaluation focused on the semantic validation of inconsistent streaming data points in each of the three temperature sensors. In the first round of the experiment, single point of inconsistent error injection occurs at every alternate second streaming window following the consistent streaming data produced by

the first temperature sensor (*Temp_Sensor1*). Similarly, the second temperature sensor (*Temp_Sensor2*) is configured to produce plausible values at every second alternate streaming window with the consistent streaming data. The third temperature sensor node (*Temp_Sensor3*) is configured to generate an arbitrary value of '8888' to represent the data points that are interpreted as missing or incomplete data, as experienced by the sensor node. The simulated value is produced at every alternate first and second streaming windows with alternate window producing the consistent streaming data. Usually, the description of missing data points is defined by the sensor instrumentation schema. Ideally, consistent values in the experiment represent true temperature readings that fall within the permitted range of values and are simulated based on intra-variability levels of comfort and standard indoor temperature recommendations.

The maximum streaming duration for each of the three temperature sensor node occurs at every 1 second, 2 seconds and 2 seconds respectively. Besides, the streaming duration for relative humidity and pressure sensor nodes are set to 7 seconds and 5 seconds respectively. The streaming duration for each of the sensors has been deliberately kept low to demonstrate the suitability of the framework to support real-time scenario with the high streaming rate (e.g location sensor), associated with tracking of a sudden change in quality requirements of the sensor streaming data. In all rounds of the experiment, inconsistent and missing streaming data points have been represented explicitly by a pseudo-value of -27.4^0 *Celsius* and '8888' respectively. Plausible values are temperature readings recorded from direct interference with external or climatic/weather temperature readings.

In the second experimental run, (*Temp_Sensor1*) was configured to produce ten (10) inconsistent data points in each affected streaming window. The streaming interval of each category of erroneous or stream quality issue is similar to the first experiment. Similarly, the streaming duration of each sensor node remains unchanged with the same pseudo-value for the representation of inconsistent and missing values during raw streaming data generation throughout the experimental run.

The semantic representation of the raw data is produced with an annotated timestamp using the domain RDF graph (derived from SmartSUM ontology) and RDF manager. The resulting RDF quadruple statement is further re-written with the RIOT API to derive the four serialisation formats discussed in 2.4.1. These serialised RDF formats consist of the RDF/XML, NTriple, Turtle and Notation Formats. The new serialised RDF formats are then subjected to further processing by the framework for the semantic validation process. The output of semantic validation was later subjected to evaluation over continuous validation cycles using the established evaluation metrics. The evaluation of the two experimental runs was conducted over 2000 cycles each.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:smartSpace="http://localhost:8080/smartSpace#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#" >
<rdf:Description rdf:about="http://localhost:8080/smartSpace#temp2Readings2">
   <smartSpace:isInconsistent>Erroneous reading</smartSpace:isInconsistent>
   <smartSpace:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#float">-27.4</smartSpace:hasValue>
   <smartSpace:tempHasTimestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2020-07-28T03:32:58.992Z
</smartSpace:tempHasTimestamp>
   <smartSpace:hasId rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Sensor 2</smartSpace:hasId>
   <smartSpace:hasSeason rdf:datatype="http://www.w3.org/2001/XMLSchema#string">summer</smartSpace:hasSeason>
   <rdf:type rdf:resource="http://localhost:8080/smartSpace#tempValue"/>
 </rdf:Description>
<rdf:Description rdf:about="http://localhost:8080/smartSpace#humitidy2Readings2">
   <smartSpace:hasHumidityReading
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">43</smartSpace:hasHumidityReading>
   <smartSpace:humidityHasTimestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2020-07-28T03:33:01.984Z
</smartSpace:humidityHasTimestamp>
   <rdf:type rdf:resource="http://localhost:8080/smartSpace#humidityValue"/>
 </rdf:Description>
<rdf:Description rdf:about="http://localhost:8080/smartSpace#pressureReading2">
   <smartSpace:hasPressureReading
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">756.75</smartSpace:hasPressureReading>
   <smartSpace:pressureHasTimestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2020-07-
28T03:33:21.986Z</smartSpace:pressureHasTimestamp>
   <rdf:type rdf:resource="http://localhost:8080/smartSpace#pressureValue"/>
 </rdf:Description>
</rdf:RDF>
```

Figure 5.6: Sample output from Semantic Validation with RDF/XML serialisation

5.6 Evaluating Effectiveness and Efficiency of SISDaV in Home Automation

The evaluation considers the metrics defined in section 5.2 to estimate both the effectiveness and efficiency of *SISDaV* framework as contextualised in the Smart Home

```
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix smartSpace: <http://localhost:8080/smartSpace#> .
smartSpace:temp2Readings2
      a                    smartSpace:tempValue ;
      smartSpace:hasId          "Sensor 2"^^xsd:string ;
      smartSpace:hasSeason       "summer"^^xsd:string ;
      smartSpace:hasValue       "15.15"^^xsd:float ;
      smartSpace:isInconsistent   "Erroneous reading" ;
      smartSpace:tempHasTimestamp  "2020-08-03T04:22:03.125Z"^^xsd:dateTime .

smartSpace:tempReadings5
      a                    smartSpace:tempValue ;
      smartSpace:errorData       "Missing Value"^^xsd:string ;
      smartSpace:hasId          "Sensor 1"^^xsd:string ;
      smartSpace:hasSeason       "summer"^^xsd:string ;
      smartSpace:hasValue       "8888.88"^^xsd:float ;
      smartSpace:tempHasTimestamp  "2020-08-03T04:22:09.154Z"^^xsd:dateTime .
smartSpace:humidityMeanValue
      a      rdfs:Class .

smartSpace:pressureMeanValue
      a      rdfs:Class .

smartSpace:pressure2Reading2
      a                    smartSpace:pressureValue ;
      smartSpace:hasPressureReading  "753.21"^^xsd:float ;
      smartSpace:pressureHasTimestamp
            "2020-08-03T04:21:40.124Z"^^xsd:dateTime .

smartSpace:humitidy2Readings2
      a                    smartSpace:humidityValue ;
      smartSpace:hasHumidityReading  40 ;
      smartSpace:humidityHasTimestamp
            "2020-08-03T04:21:40.124Z"^^xsd:dateTime .
```

Figure 5.7: Sample output from Semantic Validation with Notation3 (.n3) serialisation

case study. The number of erroneous quadruples processed by *SISDaV* in each validation cycle differs in the two experimental runs. A single completed validation Cycle is considered to be the combined window of semantic streaming data selection and reasoning in the experiment. This excludes the window containing the raw sensor streaming data generation and its translation into a logical stream or semantic triple pattern. The reasoning window (also called validation window) describes the actual inference performed on each semantic stream, based on conditions in the validation rule.

The evaluation process specifically targets estimating the semantic validation of Inconsistent data point produced within each validation window. Figure 5.6 and 5.7 show sample outputs of semantic validation of sensor streaming data in a particular validation window, produced as a triple pattern. The summary of semantic validation is produced over continuous validation windows with the cycle. Figure 5.8 indicate a typical summary of semantic validation on semantic streaming data within a validation cycle based on specific event and validation rule. In an attempt to get a smooth trend in interpreting the output of each validation cycle concerning the effectiveness and effi-

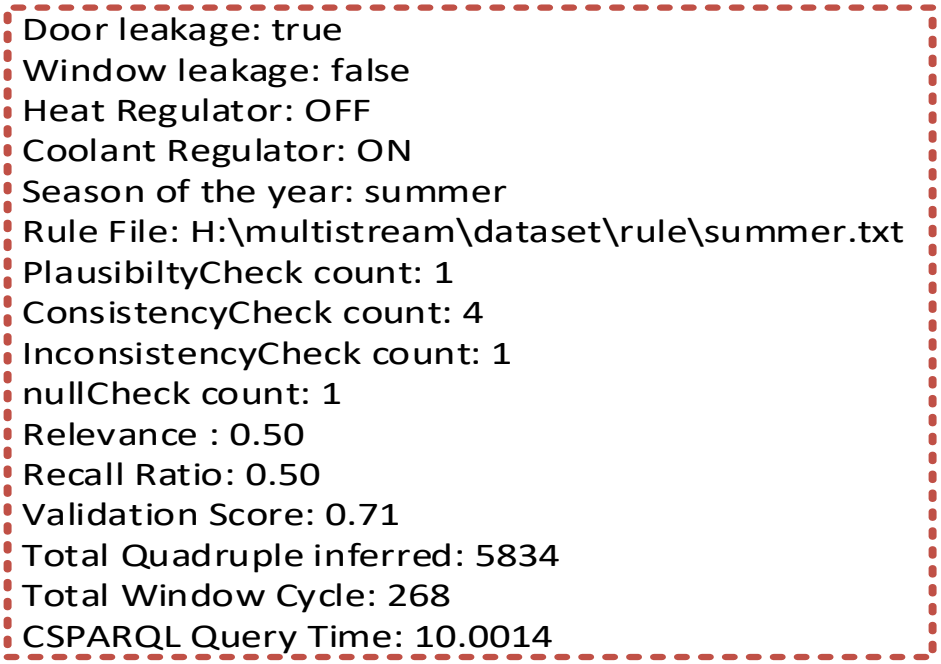

Figure 5.8: Snapshot of Output from a Validation Cycle

ciency evaluation, the Cumulative Moving Average (CMA) is applied in all the evaluation results. In the first round of the experiment involving a single point of an inconsistent data point, the semantic validation framework can perform semantic validation on average of 49 quadruples in each streaming window and a total of 3150 per validation cycle. The *SISDaV* framework can produce a total of 338,904 inferred quadruples over 2000 validation cycles.

Similarly, in the second experimental run, *SISDaV* framework is able to produce an average of 103 quadruples in each streaming window with a total of 3035 quadruples per each validation cycle. The reduction in the total quadruples processed by each validation cycle in the second experiment is due to the presence of the Hash function built as part of the framework. This is responsible for removing duplicate value with the same timestamps among raw streaming data. A total of 341,472 inferred quadruples are produced at the end of 2000 validation cycles.

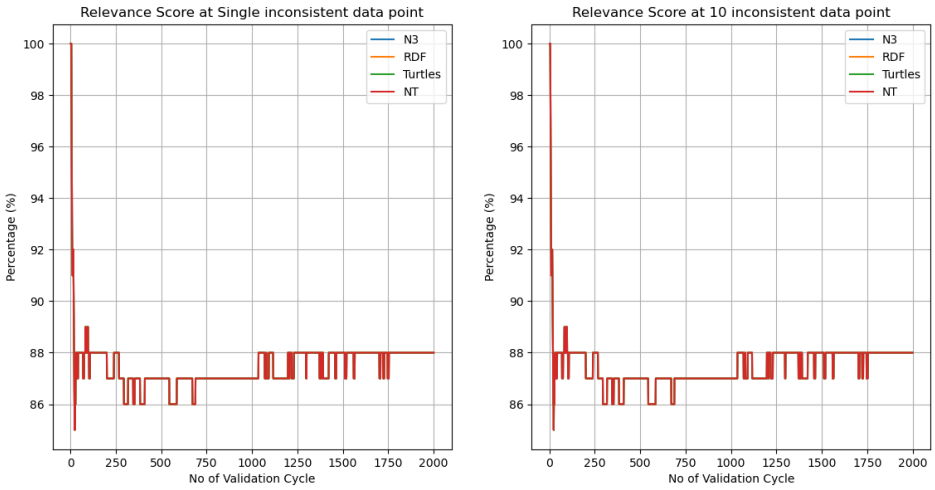

Figure 5.9: Relevance Ratio of Serialised RDF Formats at two different experimental Runs with Injections of Inconsistent Data Points per Streaming window

5.6.1 Effectiveness Evaluation of *SISDaV* in Home Automation

The two experimental runs are able to produce a total of 680,376 inferred quadruple statements (triple statement with timestamp) from the semantic validation process. The number of quadruples produced within each streaming window depends on the maximum duration of semantic stream selection and sleep duration. Figure 5.9 shows the results of the estimation of the Relevance score of the validation of *SISDaV* from both experimental runs. The score was stable between 86% and 88% for the two experimental runs with not much significant difference among the serialised formats. The spike in the first 10 windows is due to low plausibility count at the earlier stage of the streaming node. In addition, the drop in the relevance ratio between the 250th and 1000th validation cycle is caused by aggregated streaming windows with a significant number of Plausibility count. Table 5.3 and Table 5.4 provides a more detail account of the relevance ratios of the four RDF serialised formats across interval of 400 cycles. The values were reduced to range 0 to 1 and computed with Cumulative Moving Average(CMA) over successive intervals of validation cycles .

Similarly, Figure 5.10 presents the validation score showing the Validation score (accuracy) of *SISDaV* framework above 88% across all the validation cycles in both experimental run. Furthermore, the RDF serialise formats do not have any significant

Table 5.3: Relevance Score *SISDaV* with Single Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Relevance Score				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0.86	0.87	0.88	0.88	0.88
Turtle	0.86	0.87	0.88	0.88	0.88
NTriple	0.86	0.87	0.88	0.88	0.88
N3	0.89	0.87	0.88	0.88	0.88

Table 5.4: Relevance Score of *SISDaV* with Ten (10) Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Relevance Score				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0..86	0..86	0.87	0.86	0.86
Turtle	0..86	0..86	0.87	0.86	0.86
NTriple	0..86	0..86	0.87	0.86	0.86
N3	0.86	0.86	0.87	0.86	0.86

Table 5.5: Validation Score of *SISDaV* with Single Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Validation Score				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0.89	0.89	0.89	0.9	0.9
Turtle	0.89	0.89	0.89	0.89	0.89
N-Triple	0.89	0.89	0.89	0.89	0.89
N3	0.89	0.89	0.9	0.9	0.9

Table 5.6: Validation Score of *SISDaV* with Ten (10) Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Validation Score				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0.88	0.89	0.89	0.89	0.89
Turtle	0.88	0.89	0.89	0.89	0.89
N-Triple	0.88	0.89	0.89	0.89	0.89
N3	0.88	0.89	0.89	0.89	0.89

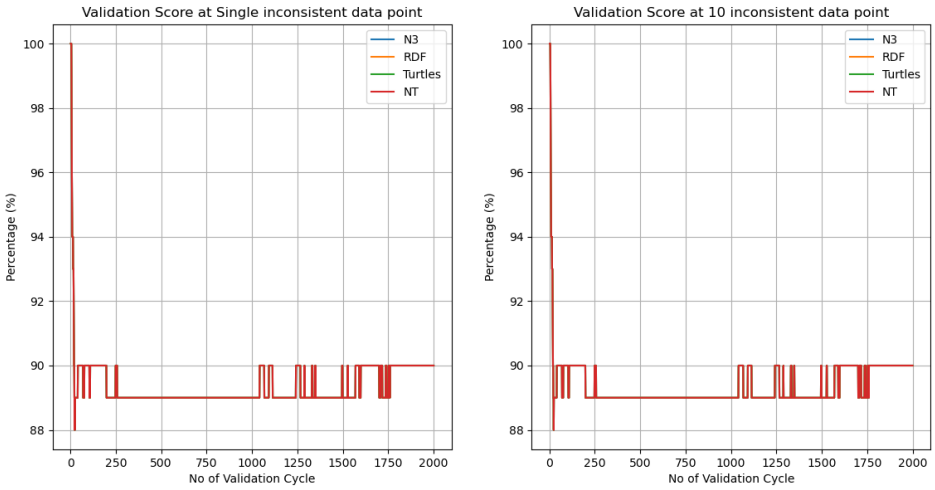

Figure 5.10: Validation Score of RDF Formats at Two Experimental runs with Injection of Inconsistent Data Points per Streaming window

effect on the semantic validation process as all of them can reach a peak of 90% in both experiments from the 10TH Cycle. Table 5.5 and Table 5.6 shows the exact value of the validation score computed as CMA at separate interval of 400 cycles for each serialised format. The results of the two experimental runs (Table 5.5 and Table 5.6) shows a very slight variation, though still insignificant.

The implication of the results from the evaluations of *SISDaV* suggests a slight change in terms of effectiveness, particularly in application with a high error rate. In addition, the result corresponds to the total fraction of the inconsistent data points injected into each validation cycle during both experiments.

5.6.2 Efficiency Evaluation of *SISDaV* in Home Automation

The efficiency of *SISDaV* has been considered in terms of Time-based performance measure. This is considered an important and relevant metric due to the dynamic nature of IoT streaming data. At first, the experimental runs computes the estimates of the average time required by *SISDaV* to complete the reasoning task for each validation cycle. Further, the evaluation also computes the estimate for both the average processing time and average latency experienced by the framework during the validation

process.

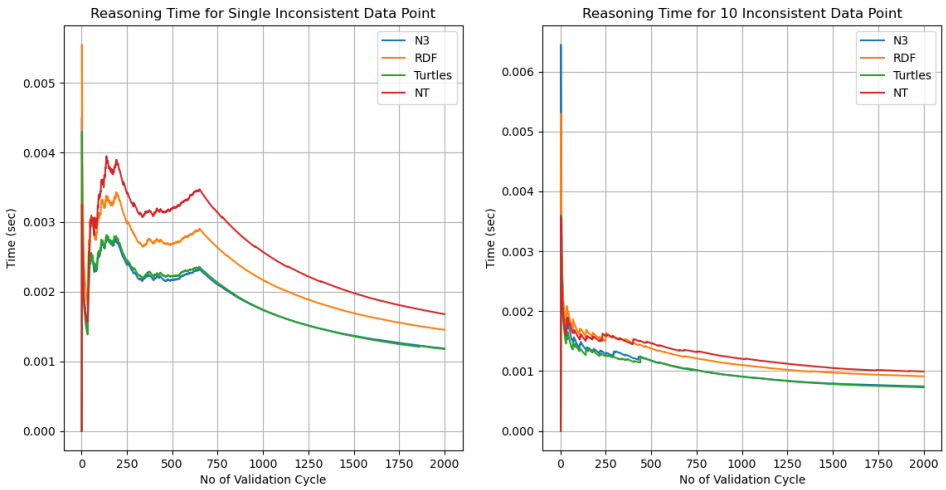

Figure 5.11: Reasoning Time of RDF Formats at Two Experimental runs with Injection of Inconsistent Data Points per Streaming window

The reasoning time, processing time and latency of the semantic validation process is estimated on each of the RDF serialised formats exclusively. Again, for each of the performance metrics, a Cumulative Moving Average of the estimates are computed over successive validation cycles in the separate experimental runs. Figure 5.11 shows the performance of the reasoner with the validation rules that were fired in each semantic validation window. The graph provides the estimate of the average time to complete a semantic reasoning task and provide an inference within a validation cycle.

In figure 5.11, N-Triple and RDF/XML formats require more time in seconds to perform inference in the first experiment, which is slightly lesser in the second experiment compared to counterpart serialised formats. The estimate from both experiments indicates the structure of N-Triple and RDF/XML serialised formats has effects on their expressivity. Most likely due to the resource-constrained feature, which will require more time to be processed by the semantic reasoner or semantic reasoning engine. Also, the speed of processing decreases along with the validation cycles for all the serialised format, thanks to the optimized matching technique embedded in the Jena2 reasoner (Carroll et al., 2004), in which the reasoning engine was built upon. More specific details of the

reasoning time across separate intervals of validation cycles are presented in Table in Appendix C.

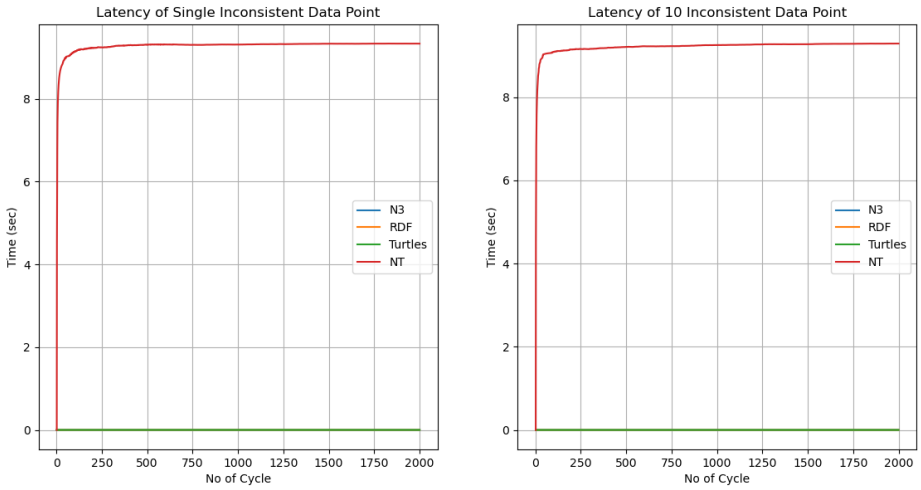

Figure 5.12: Latency of RDF Formats at Two Experimental runs with Injection of Inconsistent Data Points per Streaming window

Figure 5.12 also shows how Turtle serialised format can have a low significant influence on semantic validation process with a peak latency of 0.0023 seconds across the validation cycles as seen in both experiments. The values suggest the possibility of experiencing almost little or no delay between validation cycles in a typical smart home reactive application as contained in the use case scenario. In contrast, there is a high increase in peak latency of RDF/XML, N-Triple and N3 serialised formats. Based on evidence from the two experimental runs, the peak average latency of the three formats is measured as 9.285 seconds. The result equally confirms the effect of resource constraints on RDF data formats when it comes to the issue of performing semantic validation and inference on IoT streaming data in Smart Home automation.

Finally, the processing time estimates the combined duration for performing semantic query execution and reasoning time on every semantic streaming data within each cycle. Figure 5.13 shows no significant difference among all the four RDF serialised data formats. The peak average processing time for the RDF data formats for the two experimental runs is estimated at 9.724 seconds and 9.696 seconds respectively. More specific

Table 5.7: Average Processing Times of Serialised Formats from Two Experimental Runs

Serialised Formats	Mean Processing Time for 2000 Cycles	
	Experiment 1	Experiment 2
RDF/XML	9.733	9.710
N-Triple	9.750	9.714
N3	9.710	9.684
Turtle	9.701	9.675

detail of the average processing time of each serialised formats in the two experiments is shown in Table 5.7. Experiment 1 and Experiment 2 indicates the mean processing time of Single and Ten injections of Inconsistent data respectively. Though results of average processing time in the two experiments show a slight difference, it does suggest that enhancing semantic streaming data validation with deduplication technique can slightly reduce processing time. This is because most of the duplicated raw Inconsistent data points in the second experiment were previously pre-processed with the hashing technique.

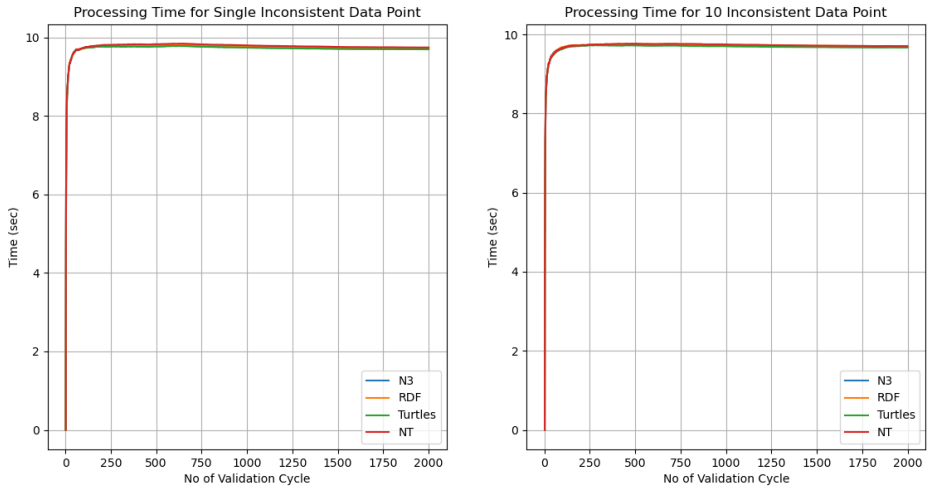

Figure 5.13: Processing Time of RDF Formats at Two Experimental runs with Injection of Inconsistent Data Points per Streaming window

5.7 Case Study 2: Smart Cities IoT-based Decision Support Systems

The case study introduced in this section describes the serious challenges in processing the quality requirements of heterogeneous sensor streaming data to effectively enhance data-driven decision support systems in the context of smart city innovations. The operational decision-making process by the city planner can focus on knowing when to reduce the Carbon monoxide concentration in the air to sustain a cleaner environment, and in attempt to realise one of the pillars of the IBM smart city model (Kehoe et al., 2011). The quality of such decisions is based on the streaming data being reported by several heterogeneous sensors deployed within the city. Based on the evidence of raw sensor data collected from a similar air quality monitoring scenario within the Italian city, it was reported the streaming data are have a number of quality issues. These quality problems include cross-sensitivities issues that are related to data plausibility and, missing values related to incomplete sensor readings. The effects of the poor estimations from sensor readings are seen to have tremendous effects in terms of poor quality of real-time decisions by the smart city planner. Generally, the real-time analysis of these heterogeneous sensor streaming data is expected to facilitate the effective monitoring, reporting, decision support and fault detection systems that support reactive services within a smart city (D'Aniello et al., 2018). The objective of the case study evaluation is to demonstrate the feasibility of the validation approach, and test both effectiveness and efficiency of SISDAV at different query windows with various streaming interval as produced at the lower granularity level of the data layer in the IBM smart city model.

5.7.1 Air Quality Data Set

In an attempt to get the view of the feasibility and evaluation of SISDaV within the context of smart cities domain, the raw data set[9] produced from embedded Air Quality

[9]https://archive.ics.uci.edu/ml/datasets/Air%2Bquality

Chemical Multi-sensor device has been considered for the experiment. The device was located on the field in a significantly polluted area, at road level, within an Italian city. The data set was collected from the Italian city between the period of March 2004 to February 2005. It contains a total of 9358 instances of multi-variate data with a timestamp. These data instances represent the averaged hourly rate from an array of 5 metal-oxide chemical sensors recordings. Specifically, the dataset consists of hourly sensor recordings represent readings for Carbon Monoxides (CO), Benzene (C_6H_6), Nitrogen oxides (NO_x), Nitrogen Dioxides (NO_2) and Titania (Non-Metallic hydro-Carbon). The description of the attributes are as follows:

- Date in DD/MM/YYYY format
- Time measured in HH.MM.SS
- CO(GT) True hourly averaged concentration CO in mg/m^3 (reference analyzer)
- PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
- NMHC(GT) True hourly averaged overall Non Metanic HydroCarbons concentration in $microg/m^3$ (reference analyzer)
- C6H6(GT) True hourly averaged Benzene concentration in $microg/m^3$ (reference analyzer)
- PT08.S2(NMHC) titania hourly averaged sensor response (nominally NMHC targeted)
- NO_x True hourly averaged NO_x concentration in ppb (reference analyzer)
- PT08.S3(NOx) tungsten oxide hourly averaged sensor response (nominally NO_x targeted)
- NO2 True hourly averaged NO2 concentration in $microg/m^3$ (reference analyzer)
- PT08.S4(NO2) tungsten oxide hourly averaged sensor response (nominally NO2 targeted)

- PT08.S5(O3) indium oxide hourly averaged sensor response (nominally O3 targeted)
- T Temperature in Â°C
- H Relative Humidity (%)
- AH Absolute Humidity

Some of the data points in the entire data set are reported to contain issues relating to cross-sensitives/inference (Plausibility) problem, sensor and concept drift, and Missing data points (Incompleteness problem). Specifically, the missing data points from the sensor readings are recorded as -200, while cross-sensitivity issues are considered as sensor readings that interfered with other related phenomena. The subset of raw data set of interest to the study can be found in Appendix C.

5.7.2 Experiments

The aim of this experiment with the focus case study is to evaluate the effects of raw data streaming rate and semantic query time on both effectiveness and efficiency of the approach. The experimentation process has been conducted by re-using the standard testbed for the smart home case study with similar hardware and software requirements (refer to 5.5.2). Two separate rounds of experiments were conducted for each duration of 72 hours and 120 hours respectively. The streaming windows and the query execution duration were varied to subject the framework to stress and different experimental conditions. Each round of the experiment concurrently and explicitly performs the semantic validation process using the serialised RDF formats with the same raw streaming data at run-time. The process is repeated for the different experimental conditions involving the streaming windows and semantic query duration.

The streaming windows and generation for each sensor data values with timestamps relating to CO, NO_x, NO_2 and C_6H_6 readings were simulated based on the values of data set from the air quality measurement as discussed in 5.7.1. The run-time simulated

streaming data values are required for the activation of the real-time streaming data quality validation process. During the streaming data generation, the poor quality attributes comprising of incompleteness, inconsistency and plausibility were injected at specific streaming windows as contained in the actual air quality dataset. Specifically, missing or incomplete sensor stream values are injected as -200, inconsistent/redundant sensor stream are values outside the range 0.1 and 11.9 (being the actual minimum and maximum values collected from CO sensors for the period of the project), and plausible values are CO values that exhibit cross-sensitivity with that of C_6H_6 values at the same timestamp. The assumption to base the relationship of false positive (plausible) values on this combination is because the porosity of hydrogen makes it possible to interfere with CO sensors thereby interfering with the measurement (Chao et al., 2005).

To allow better understanding and proper evaluation of the validation framework (*SIS-DaV*), erroneous data points are bound to the raw CO reading, which was maintained at fixed streaming window interval for a fixed number of cycles in both rounds of experiments. Each cycle contains a total of ten (10) continuous sensor streaming values which represents the consistent, plausible, incomplete and Inconsistent streaming data.

Specifically, the first five streaming windows (i.e windows 1 to 5) of the CO sensor produces incomplete (missing data) data, the next three windows (6th to 8th) produces the inconsistent streaming values, the 9th and 10th streaming windows produces consistent and plausible values respectively. In the first round of the experiment, all the four types of sensors are allowed to produce streaming data simultaneously at every 1 minute with the query execution time being kept at every 10 minutes with 2 minutes sleep time between successive query execution. The streaming and query execution duration was adjusted in the second round of the experiment. This time, the streaming window for each of the four types of sensors is set to 4 minutes with the query execution time of 20 minutes at an interval of 20 minutes between successive semantic query. These raw streaming data produced from the physical sensors deployed within the city

is converted into appropriate serialised RDF data to allow semantic stream selection to take place.

5.7.2.1 Semantic Stream Selection

The stream selection technique is similar to the one in the Smart Home Automation case study, which uses the C-SPARQL query as the semantic stream aggregation or collection technique. In the current case study, the stream selection duration for the query execution at two different situations with different intervals was considered. The first stream query shown in figure 5.14 performs the stream selection for a duration of 10 minutes with a maximum sleep time of 2 minutes between successive query execution. This will allow the semantic query to collect all the semantic streaming data in the previous consecutive windows for 10 minutes with an interval of 2 minutes between the next query execution. The stream selection duration was adjusted in the second round of the experiment to reflect 20minutes for query execution or selection duration with a maximum sleep of 20 minutes between an interval of query execution. The duration for query execution caters for an extreme case of reduced duration for sensor and IoT streaming data generation within the Smart City context, which will normally be configured at an interval of 30 minutes or more. The semantic query is able to process each of the heterogeneous streaming data concurrently by using the graph pattern matching readily available within the C-SPARQL library. The output of the semantic stream selection is pushed to the semantic reasoning engine, which performs the validation process in a cycle pattern by utilising the validation rules.

5.7.2.2 validation Rules

The previous study (Roberge, 2000) that involves air quality measurements has shown the relationship that exist between Carbon Monoxide (CO) and Nitrogen Oxides (NO_x)/ Nitrogen Dioxides (NO_2) are inversely proportional, while the relationship between (NO_x) and (NO_2) are directly proportional. Based on the findings from the study, it means that a reduction in the amount of CO emission will cause an increase in (NO_x) emission. Therefore, it makes sense to establish the validation rules for CO readings

```
1     REGISTER QUERY sensorValueOf AS
2.    PREFIX smartSpace:<http://localhost:8080/smartSpace#>
3.    SELECT *
4.    FROM STREAM <http://localhost:8080/smartSpace/streamCO> [RANGE 10m STEP 5m]
5.    FROM STREAM <http://localhost:8080/smartSpace/streamNOX> [RANGE 10m STEP 5m]
6.    FROM STREAM <http://localhost:8080/smartSpace/streamNO2> [RANGE 10m STEP 5m]
7.    FROM STREAM <http://localhost:8080/smartSpace/streamBenzene> [RANGE 10m STEP 5m]
8.    WHERE
9.    {
11.         ?COReadings smartSpace:hasCOValue ?COVal.
12.         ?COReadings smartSpace:COHasTimestamp ?COTime.
13.         ?NOXReadings smartSpace:hasNOxValue ?NOXVal.
14.         ?NOXReadings smartSpace:NOxHasTimestamp ?NOXTime.
15.         ?NO2Readings smartSpace:hasNO2Value ?NO2Val.
16.         ?NO2Readings smartSpace:NO2HasTimestamp ?NO2Time.
17.         ?benzeneReadings smartSpace:hasBenzeneValue ?benzeneVal.
18.         ?benzeneReadings smartSpace:benzeneHasTimestamp ?benzeneTime.
19.   }
20.   ORDER BY ASC(?COTime)
```

Figure 5.14: Semantic Stream Query listing

based on these relationships.

Based on the above-established relationships, a sample validation rule in figure 5.15 is developed for checking the inconsistent semantic streams produced from the carbon monoxide sensor. The rule specifically compares the current CO reading with a possible range of consistent values with the timestamp, which must have been pre-determined by the city planner or certified safe level of exposure. Being a class of heuristics, the rule also relates CO reading with that of Benzene readings obtained within the same timestamps to define the Plausibility rule (Figure 5.16). The Incompleteness rule is able to track CO readings with an exact value of -200 and which corresponds to the same value of NO_x in the previous or current timestamp.

The three rules are processed by the semantic reasoning engine in a sequential manner. In a more practical sense, each semantic streaming data previously aggregated by the semantic query is first executed against the stream completeness rule to check if object value of the semantic stream is complete or not empty, based on the schema matching.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#

[inconsistentCheck:
        (?COReadings smartSpace:hasCOValue ?COValue)
        (?COReadings smartSpace:COHasTimestamp ?COTime)
        greaterThan(?COValue,11.9)
        (?NOXReadings smartSpace:hasNOxValue ?NOxValue)
        (?NOXReadings smartSpace:NOxHasTimestamp ?NOxTime)
        greaterThan(?NOxValue,322)
        lessThan(?NOxValue,2683)
        (?benzeneReadings smartSpace:hasBenzeneValue ?benzeneValue)
        (?benzeneReadings smartSpace:benzeneHasTimestamp ?benzeneTime)
        notEqual(?COValue,?benzeneValue)
        equal(?COTime,?NOxTime)
        equal(?COTime,?benzeneTime)
        ->
        (?COReadings smartSpace:isInconsistent 'Inconsistent value')
]
```

Figure 5.15: Validation Rule for Inconsistent Carbon Monoxide Readings

The reasoner executes the plausible rule against the semantic streams in the next step once the triple attributes of the stream are complete. In case the output of plausibility rule is false, the consistency rule is then executed in the third step of the execution plan to validate the object value of the triple is still within the true range of values. The reasoning process immediately terminates on the current stream once a specific rule condition in the current validation process is satisfied. Inference produced as an output of the validation process is released for use by IoT-based reactive applications or to support smart city decision support system through a web-based interface in a near real-time manner.

5.7.3 Results and Analysis of case Study 2

In order to analyse the outputs from the semantic validation framework, the metrics described in section 5.2 is used to further estimate the effectiveness of the framework.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#

[plausibilityCheck:
        (?COReadings smartSpace:hasCOValue ?COValue)
        (?COReadings smartSpace:COHasTimestamp ?COTime)
        greaterThan(?COValue,0.1)
        (?benzeneReadings smartSpace:benzeneHasTimestamp ?benzeneTime)
        (?benzeneReadings smartSpace:hasBenzeneValue ?benzeneValue)
        greaterThan(?benzeneValue,0.1)
        lessThan(?benzeneValue,63.7)
        equal(?COValue,?benzeneValue)
        equal(?COTime,?benzeneTime)
        ->
        (?tempReadings smartSpace:isPlausible 'Plausible Value')
]
```

Figure 5.16: Validation Rule for Plausible Carbon Monoxide Readings

Similarly, the performance is based on a metric that involves reasoning time, latency and processing time as applied in a smart city case study. The analysis of the semantic validation process from both experiments produced a total of 57,016 inferred quadruples in the two experimental runs. For the first round of the experiment, each validation cycle processed an average of 36 quadruples while the second experiment was only able 20 quadruples per validation cycle.

Furthermore, figure 5.17 and figure 5.18 describes the effectiveness of the approach based on the different streaming window and stream selection duration. The Relevance score (Figure 5.17) shows no clear difference among all the RDF serialisation formats. The relevance score for each validation cycle and the streaming window is stable between 98% and 99% across 350 validation cycles. The variation in the pre-

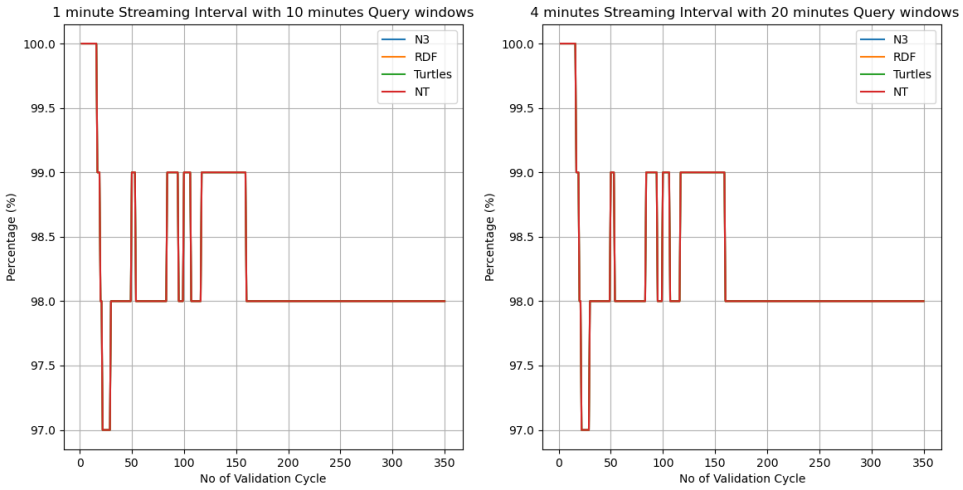

Figure 5.17: Relevance Scores from Separate Streaming Windows and Query Execution Interval

ciseness of the validation process between the first 50 cycle is due to the initialisation of the streamers at the early stage streaming window. Similarly, the validation score (Figure 5.18) of the RDF serialised formats shows the accuracy of the validation process to lie between 87% and 88%. The margin of almost 12% in the result of the validation score shows the region in the validation process where the streaming windows are that contains traces of missing and plausible data points.

In terms of estimating the reasoning time of the four serialised formats. Figure 5.19 shows that varying the query time with interval sleep time plays a significant role in the semantic reasoning process. The first experiment shows N-Triple is fairly expensive in terms of reasoning time with short query duration when compared to other formats. This is reasonably due to its constrained expressivity during processing by the semantic reasoner. The performance of N-Triple from the first experiment indicate it will require more time to complete a reasoning task, relative to the counterpart formats. The reasoning time of N-Triple initially improved exponentially compared to RDF/XML, within the first 300 cycles in the second experiment with values between 0.03 seconds and 0.044 seconds before gradually increasing. In addition, the RDF/XML format is considered another expensive format based on high streaming and query execution time in the second experiment. The poor performance of RDF/XML can be trace to the

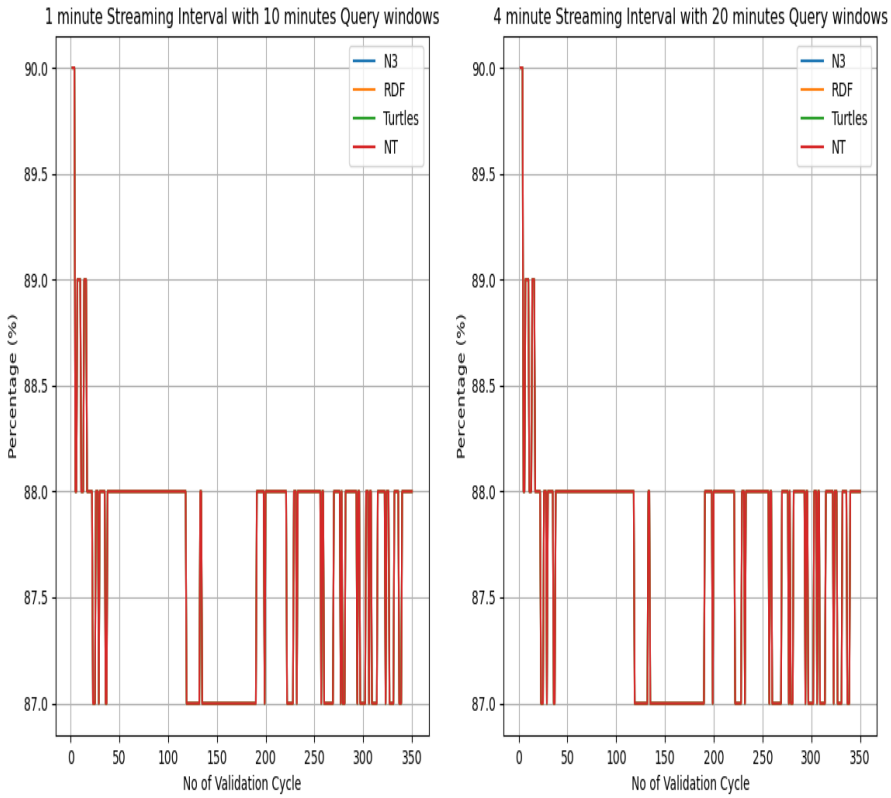

Figure 5.18: Validation Scores from Separate Streaming Windows and Query Execution Interval

verbose or wordy format, which will require more read time by the reasoning engine when performing semantic inference.

The results of the effectiveness and efficiency evaluation of the validation framework focuses on the detection of inconsistent data points produced from the CO sensor in each streaming cycle. There is no significant difference among the four serialised formats in terms of Processing Time as shown in figure 5.20. Despite the different experimental conditions in which the semantic validation process was conducted, the total duration required by the framework to perform the semantic query and reasoning on each validation cycle is stable at 600 secs beginning from the first 20 cycles from both experimental runs.

Similarly, the Latency estimates maximum delay measured in seconds between two successive validation cycles experienced by the validation framework. Figure 5.21

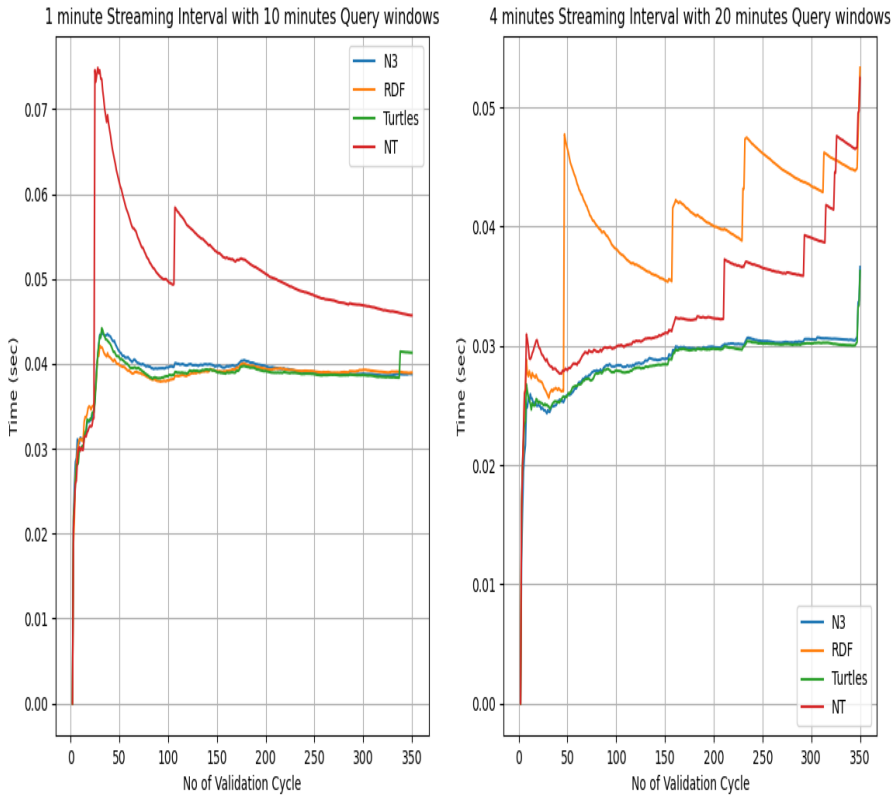

Figure 5.19: Reasoning Time ofTwo Experimental Runs with Different Conditions

indicate there is no significant difference in the latency of RDF/XML. N3 and N-Triple formats. This is apparent as the three different formats experienced the same duration of delay across the validation cycles. The streaming time and query execution/sleep time also has a significant influence on the latency of the validation framework. On the contrary, both experimental runs with different streaming and query duration show that the Turtle serialised format does not experience any delay between intervals of validation windows across the validation windows.

5.8 Discussion of Results

In general, enhancing the IoT streaming data with semantics is a step to revolutionize the performance of IoT-based reactive services and data-driven decision-making processes for relevant web-based applications. The evaluation considered the semantics of RDF data formats on effectiveness and efficiency. The efficiency evaluation has been

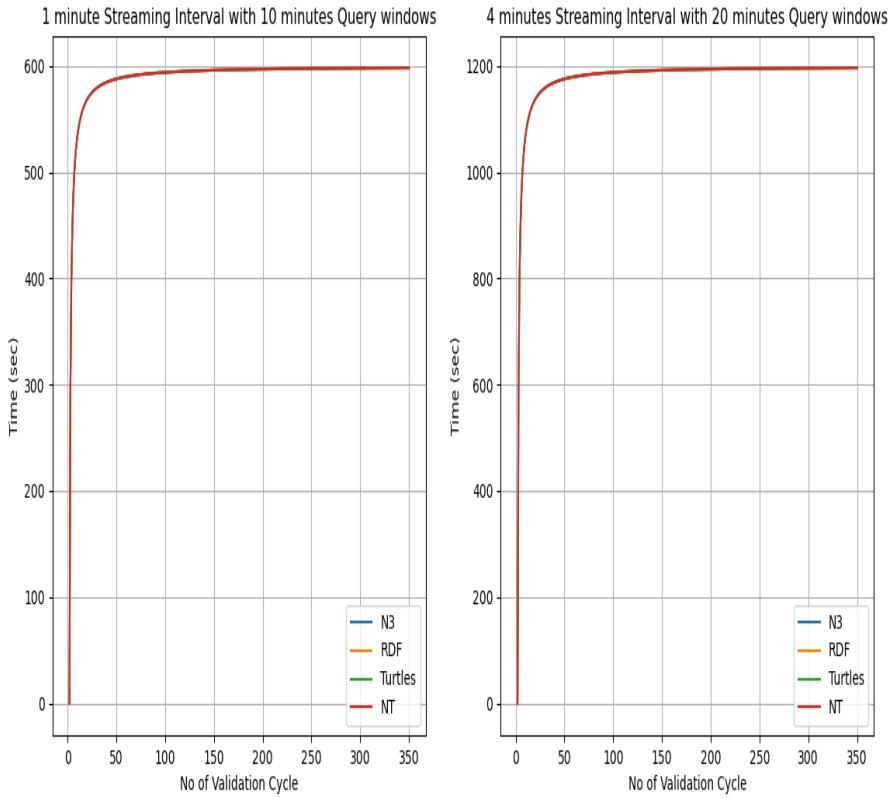

Figure 5.20: Processing Time of Two Experimental Runs with Different Conditions

estimated in terms of time-based requirement on expressivity and resource consumption (i.e time requirement) during the process of semantic validation of IoT streaming data.The evaluation with application case study considered the case of sensor streaming data that supports a Smart Home automation and the Smart City decision support System. The two case studies described how the quality of the streaming data can easily influence the output or performance of IoT-based reactive applications and data-driven decision support system respectively.

In an attempt to properly evaluate the effectiveness and efficiency of the proposed framework, it was subjected to a reasonable level stress test and comparative evaluation. By this, the case study experiments were subjected to experimental conditions that include varying the error injections rate and streaming rate in the first case study (section 5.3), and varying the semantic query time with the data streaming rate in

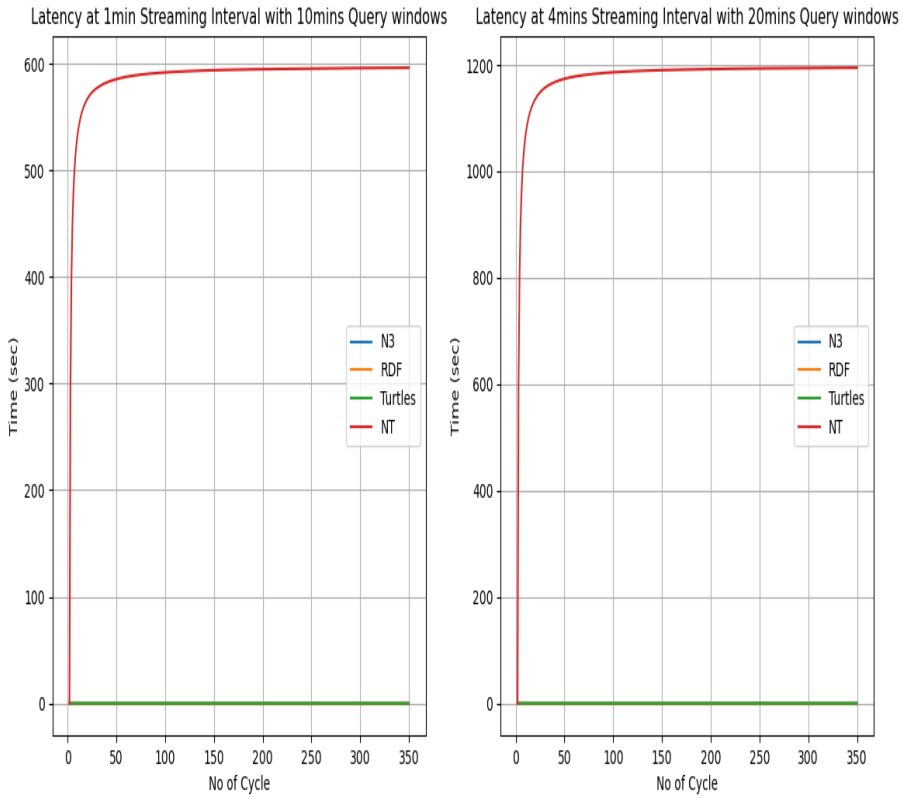

Figure 5.21: Latency of Two Experimental Runs with Different Conditions

the second case study (5.7). The stress test support an understanding of the strength of the framework in extreme conditions of faulty IoT/sensor nodes or poor quality streaming data. In the two application case studies, the evaluation of the framework follows a comparative method that is based on the comparison of effectiveness and efficiency among the various RDF serialisation formats. Specifically, the effectiveness and efficiency evaluation of *SISDaV* in dealing with the quality requirements of sensor streaming data with a focus on inconsistency, incompleteness and plausibility of the sensor readings at run-time. The effectiveness evaluation in both use cases proves that the validation technique can be sustained in Spatio-temporal software applications. This is evident from its ability to manage the varying amount of raw streaming data using window-based processing. Throughout the evaluation of both case studies, the background knowledge that has been modelled as SmartSUM ontology have facilitated the effective pre-processing raw IoT streaming data and semantic IoT stream persistent.

The validation of both experiments from the two case studies was conducted by implementing a program to determine the total no of erroneous data points that are separately produced by each streaming thread within each streaming window. The result was compared to the total number of detection points tracked by SISDaV in each cycle. The distribution of the three categories of IoT stream quality issues are identified and compared prior and after the semantic validation process for every streaming windows. The results of each experiment from the two case studies were subjected further analysis to enhance its interpretations by using the Cumulative Moving Average (CMA). The CMA provides a proper way of smoothing and computing the running average of the ordered evaluation results (computed based on effectiveness and efficiency metrics) from each validation cycle. This allows filtering of all possible noise in the pattern and to present the trend on the graph in a clear manner

The result of performance from the experiments clearly indicates there is no difference in the pattern of RDF data formats in latency and processing time from experiments in both case studies. Results from both application case studies show that the semantic reasoning time within the framework can be contained in the interval between reads of sensor nodes. Also, the efficiency is able to scale reasonably with an increasing number of validation cycles in both experiments. This means the framework is able to properly validate every streaming data points in a big sensor data environment that permits sufficient semantic query time with reasonable time in delay. The accuracy of the validation process in both case studies is reasonably stable between eighty-five and ninety-three percent. This is due to randomisation in the streaming data generation during run-time validation. In a real-life practical application, it is expected that the accuracy of the validation will lie between 85 and 100 percent based on the outcome of the experiment and standard operating condition.

Furthermore, in terms of optimising the performance of the reasoning process for time efficiency, RDF suggests being slightly expensive compared to the other serial-

isation formats. In addition, RDF data serialisation is clearly shown to be constrained by system resources such as CPU time and Operating system job scheduler. Similarly, estimating the efficiency of the framework from the dimension of latency and processing time does not really have any visible difference, except it may be reasonable to consider the Turtle format in cases where latency has a significant impact on semantic processing. Hopefully, these preliminary findings can help the understanding of most suitable semantic data serialization format with RDF data, which has been an issue in implementing a semantic stream processing system.

In general, the evaluation outcomes shows the feasibility of the semantic validation framework and also indicate that the semantic validation approach is time-based. The results from the experiment indicate that the time-based requirement for the completion of the validation process varies proportionately with the experimental conditions. This confirms the extent of the effectiveness and efficiency of the framework in related smart space and IoT-based application scenarios.

5.8.1 Comparison with Similar Semantic IoT Streaming data processing Approaches

Due to the early and emerging stage of research in the application of semantic approach in IoT streaming data validation, only few approaches shares similar aspects with this work. Firstly, the evaluation of SISDaV is performed to determine its feasibility as a domain-agnostic solution that can serve as complementary to RDF stream processing systems such as C-SPARQL, CEQLS, ETALIS and EP-SPARQL, which are currently lacking in terms of reasoning capabilities. Secondly, in comparison to the existing smart city Semantic-Driven framework (D'Aniello et al., 2018) and the C-SWRL IoT stream approach (Jajaga & Ahmedi, 2017), SISDaV provides effective and efficient IoT streaming data processing at lower a abstraction level rather than the context level as demonstrated by the both frameworks.

Similar to SISDaV, the smart city Semantic-Driven framework uses the C-SPARQL to aggregate RDF streams that was previously annotated with existing SSN ontology and domain models. However, their framework is not proven to be capable of handling a concurrent heterogeneous multi-stream aggregation tasks for real-time data interoperability. Besides, the SSN ontology used for the data stream enrichment and annotation is incomplete and ambiguous in terms of concept description. It also filters and aggregate raw streams directly from sensor nodes at higher level abstraction and send them to the next upper layer. However, SISDaV adopts the modified SSN ontology with hash function and matching techniques to support multi-stream run-time annotation and aggregation of IoT streaming data during IoT stream processing and validation. It also emphasize on implementing the approach at lower level of abstraction before it can be utilised at higher level of abstraction. In terms of reasoning approach, the smart city framework adopts implement Trowl reasoner for its reasoning system. The reasoning system does not support the closed-world assumption whereas, SISDaV was able to leverage the Jena reasoning subsystem to maintain the closed-world assumptions. In addition, the overall reasoning time takes significantly longer time when compared to SISDaV. However, both approach conforms to previous work (Tallevi-Diotallevi et al., 2013) that confirms smart city applications supports decision-making systems that utilises time-based window processing.

Furthermore, Compared to C-SWRL, SISDaV have also implemented a layered semantic framework that adopts C-SPARQL with RDF data model and semantic rule reasoning at the center of the semantic approach. It provides a feature to evaluate the alternative serialised RDF data formats to gain the understanding of the expressivity at lower abstraction level. Also, SISDaV approach considers missing data as a type of sensor stream quality issues. In C-SWRL this was treated as Negation-as-Failure using the C-sPARQL library, which was limited in tracking various forms of incomplete stream and other types of quality issues in IoT streams. SISDaV have effectively and efficiently supported all forms of incomplete IoT streams including other types of quality issues associated with IoT streaming data by providing the generic Time-aware validation rules.

Finally, the evaluation of SISDaV approach and its framework have demonstrated both effectiveness and efficiency of the approach in time-critical reactive applications involving the IoT streaming data. It also provides a window-based processing support with its continuous rule and reasoning features, which are currently missing in the smart city Semantic-Driven and C-SWRL approaches.

5.9 Conclusion

In the Chapter, the evaluation of SISDaV have been conducted in two ways: to determine the effectiveness in terms of the ability to properly identify poor quality run-time IoT streaming data and, the efficiency by being able to operate in a timely manner. Both evaluation adequately shows promising results with provision for improvement in the future.

The RDF serialised data formats have been evaluated for representing and reasoning of IoT streaming data. Various types of sensor streaming data have been analysed and compared using the different RDF serialised formats with a focus on effectiveness and efficiency in terms of semantic query duration, types of stream quality issues and, number of validation cycles. It is observed that the expressivity of semantic data format can play some role in the aspect of semantic reasoning on run-time IoT streaming data validation. The experiments further confirm that semantic technology including standards are compatible with IoT streaming data processing. It enhances interoperability and real-time reasoning in an IoT streaming environment with relatively large data size.

The experimental outcomes demonstrate the feasibility of the proposed semantic IoT Streaming Data Validation Framework (*SISDaV*), as it applies to IoT-based reactive services/applications and data-driven decision-making systems. The evaluation is able to show the extent of the effectiveness and efficiency of *SISDaV* in the context of smart home automation and smart city data-driven decision-making activity. Furthermore,

evaluation of *SISDaV* framework clearly achieves the focus of the fourth objective of the study. Therefore, it can be considered that *SISDaV* framework possess some degree of agility as its components can interact in an iterative and timely manner to deliver the output of the validation process.

Sixth Chapter

Conclusions and Future work

The research conducted in this thesis is based on the application and enhancement of semantic stream reasoning domain, which main focused is on integrating the RDF stream processing system with reasoning technique for IoT streaming data. The findings from the research has the potential for filling the gap between the IoT paradigm where streaming data are often generated and processed, but with no standard format or semantic representation and reasoning abilities.

The primary aim of this research is to develop a semantic-driven approach and its resultant framework that integrates a set of proposed new models and mechanisms including semantic sensor ontology extensions, data transformation model, enhancement of Jena rule syntax, generic sensor streaming data validation rules and the incremental IoT data persistent model. Collectively, these models and mechanism serve to provide effective and efficient IoT streaming data pre-processing, semantic modelling, selection and reasoning to achieve run-time data stream quality validation tasks. The remaining part of this chapter discusses how the objectives defined in the first chapter are met. This is followed by research contributions. The future direction of the research is outlined in the final section.

6.1 Analysis of Research Objectives

In the thesis, several design requirements were identified that are relevant to the major aim of this research (as outlined in Chapter one) :

"To Provide an Effective and Efficient Approach for Semantic Validation of IoT streaming data at run-time".

These requirements include Time consideration, data integration, data-driven processing, semantic query support, Inference support, Stream quality management, and Historical stream management. The major objective of the research is to develop *SISDaV* - a semantic-driven framework for quality validation of RDF-based IoT streaming data. *SISDaV* is based on semantic modelling, querying and continuous reasoning with a set of time-aware validation rules. Table 6.1 shows how *SISDaV* framework is able to satisfy the requirements with the design considerations and features. The framework is able to achieve both Time consideration (R1) and integration (R2) requirements through the feature of data stream selection, continuous Time-aware rules and semantic reasoning. Data-Driven Processing (R3) is considered to be the essential aspect of Data Transformation and Stream selection features with C-SPARQL driving the multiple query aspect (R4). *SISDaV* achieves stream quality management (R6) with the support of the RDF stream processing through the injection of semantic rules into the stream processing pipeline to perform semantic reasoning for inference support (R5). Incremental persistence feature facilitates newly inferred outputs to be added to the background knowledge graph through node join operation as part of historical data processing (R7).

6.1.1 Objective I: Relationship of Sensor and IoT Stream quality Problem

In the time past, various classifications have been provided for describing the common data quality issues in sensor readings. The first objective of the thesis was to review various literature to see the common trends and define a relationship between the

Table 6.1: SISDaV Features and Design Requirements

Requirements / Design Decisions	R1	R2	R3	R4	R5	R6	R7
Data Transformation			*				
Multiple Data Selection	*	*	*	*			
SenTAR	*	*			*	*	
Continuous Reasoning	*	*			*	*	
Incremental Persistence							*

various sensor data quality problems in relation to IoT streaming data. By accomplishing this objective, taxonomy of Uncertainty problem in sensor/IoT streaming data have been provided (described in section 2.2). This enabled the development of a foundation for developing insight into providing a holistic approach to solve the quality issues in an Internet of Things environment, where data are being produced in high volume with associated velocity.

6.1.2 Objective II: Evolving and Lightweight Ontology model for IOT Resource and Sensor Streams Specification and Integration

The second objective was to develop a model that can: (i) homogenize different IoT nodes to achieve interoperability (ii) assist in the annotation and integration of IoT streaming data (iii) support the RDF serialised data format for IoT streaming data. The objective has been accomplished by extending related ontology models and adapting relevant schema for the successful delivery of the IoT streaming data validation system requirements. The development and implementation of SmartSUM Ontology achieved this objective (as described in section 3.3). Being lightweight, SmartSUM consist of simple unambiguous specification and reusable concepts that can be easily transformed into RDF graph. Its evolving attribute makes it possible to support the frequent update of new inferred streams and addition of new IoT node.

6.1.3 Objective III: Continuous Reasoning Approach with Generic Data Validation Rules

The third objective is to enable the continuous reasoning technique that facilitates the run-time inference task for IoT streaming data during the stream validation process. The objective has been accomplished through the use of modified Jena rule language with reasoning subsystem (refer to section 4.3.3). The run-time requirement of the continuous reasoning engine benefits from the temporal component of C-SPARQL query that was layered with the new time-based Jena rule syntax. Specifically, the continuous feature of the reasoning engine combines C-SPARQL with the time-aware rules in the native Jena reasoner to perform the continuous reasoning and inference task.

6.1.4 Objective IV: Approach Integration and Automation into a Tightly-Coupled Unified Framework

The fourth object focused on achieving a unified tightly-coupled semantic-driven framework for the processing and validation of IoT streaming data at the semantic layer of IoT applications. This objective has been well accomplished with *SISDaV* framework (refer to section 4.4). *SISDaV* can combine the task of RDF data serialisation with the semantic stream processing and continuous reasoning process while relying on the rule selection algorithm to perform the semantic data stream validation task. Each process is performed sequentially and exclusively until inference is produced and the incremental data persistence is completed. The framework is, therefore, able to facilitate run-time semantic integration and interoperability, stream query with reasoning support, and maintenance of the background knowledge graph for consistency of concepts during the validation process of IoT streaming data.

6.1.5 Objective V: Evaluation using Real world Case Studies with focus on RDF Serialised Formats

To enable critical evaluation process, the thesis considered different real-life case studies with several rounds of experiments using different real-life data sets from different IoT/Sensor streaming domains. Firstly, several sensor stream quality issues were researched and investigated to enable the construction of the prototype architecture (see Section 5.4). Considering the common quality issues relating to Inconsistency, plausibility and Incompleteness/Missing problems in sensor/IoT streaming data, high volume data dataset from smart home automation and air quality measurement in smart city projects are processed for streaming data quality validation (refer to sections 5.3.1 and 5.7.1). Secondly, experimental runs for a specific case study concerning each of the quality problems was performed under various experimental conditions (refer to sections 5.5.2 and 5.7.2). The implementation of the prototype architecture also demonstrates the feasibility of the approach. Thirdly, the comparative evaluation of the semantic validation process with the semantic IoT data stream model were performed in sections 5.6 and 5.7.3. The outcomes show that the proposed approach can achieve both the effectiveness and efficiency of the approach.

In particular, to evaluate the effectiveness and efficiency of the approach, a comparative method of the serialised RDF data is performed with four different real-life application situations to determine the Relevance and Validation scores, semantic reasoning time,processing time, and Latency (refer to sections 5.6 and 5.7.3). These suggest that the proposed semantic-based approach enhance the process semantic IoT streaming data serialisation, selection, reasoning, and quality validation via a common Application Programming Interface.

6.2 Conclusions and Contributions

The processing of raw IoT streaming data has always focused on the application of semantic process at a higher level of applications. The findings from the investigations have demonstrated that semantic process is possible at the lower granularity levels of IoT processing applications, which can include both the physical and logical representations of the IoT streaming data. In particular, the work demonstrates the semantic serialisations of the raw IoT streaming data using the alternative formats for RDF data instead of the well known or popular XML-based format used in many applications.

While several reactive IoT applications and event processing systems focused specifically on semantic modelling of sensor streams and detection of events from these streams, the continuous validation rules embedded within the *SISDaV* framework is concerned with continuous validation of sensor streaming data and invocation of action in response to specific IoT streaming data quality requirements. This feature is also able to effectively and efficiently performs IoT streaming data quality detection and mitigation in a near real-time pattern. The following contributions have been added to the body of knowledge during the study.

6.2.1 Contribution I: SmartSUM Ontological Model

This thesis has presented a generic, lightweight semantic model named SmartSUM. It builds on the major existing semantic sensor models, data quality dimensions, IoT domain resources, Observation and Measurement schema, Time ontology, and SOSA ontology model. It possesses two main features: 1) it support the evolution of knowledge graph based on the incremental data persistence and dynamic nature of the streaming data and IoT validation process; 2) Knowledge graph embedding feature with consistency checking among newly added concepts or individuals representing the IoT streaming data.

In addition, the model allows easier re-usability in other related domain due to its unambiguous specification of concepts including its relevance in a similar streaming environment where data plays a major role in driving processes for reactive services.

6.2.2 Contribution II: SenTAR Rules with Continuous Reasoning Approach

This thesis proposed a generic Semantic Time-Aware Validation Rules called *SenTAR* to enable the definition of the validation rules for Inconsistent, Plausible and Incompleteness IoT streaming data. The rule is developed as an extension of the popular Jena rule syntax, which is processed as a forward chain production rule. The Addition of new window constructs enables users to specify the time constraints to be used by the native validation or processing engine as part of the software project. These time requirement can be defined at the rule level, which executes globally across all the streaming windows in the IoT streaming environment, or at a local window that is restricted to a specific application. SenTAR complements C-SPARQL and existing Jena reasoning engine or subsystem to achieve continuous reasoning functionality.

6.2.3 Contribution III: SISDaV framework

Based on the aforementioned semantic-based system, SISDaV was implemented. It incorporates various mechanisms (such as the raw data transformation, selection algorithm and serialisation) with the semantic process (including querying and reasoning) to form a layered structure. The mechanisms collectively describes the semantic validation approach and also forms the fundamental building block for SISDaV.

SISDaV is capable of managing IoT streaming data at the granular level of the web applications, in order to provide data validation operations to reactive services and related data-driven decision making systems. In addition, owing to its semantic-driven technique, the validation approach also manages the semantic stream selection and continuous reasoning with rules as a single unified system. It can provide automated

services to support the performance of web-based applications in the internet of things environment.

6.3 Future work

This research is focused on enhancing rule-based reasoning to support typical RDF stream processing system towards achieving a semantic validation approach for quality requirements in IoT streaming data. One of the other major areas that contribute to stream quality problems in sensors is the security violation of IoT in a cyber-physical environment. This is because recent findings indicate that when IoT components are attacked, the behaviour of such components can deviate from the expected pattern which can result in sensor providing false or poor quality readings. In future work, the validation framework will consider including privacy management as part of the extended solution to support such a challenging environment.

In addition, for the future development of the semantic reasoning engine, the existing research will need to be extended to support a distributed reasoning approach. The option to decentralise the current reasoning approach will considerably improve the scalability of semantic streaming data validation process, especially in a high computing environment where constraints on resource usage are highly predominant and inevitable.

In the meantime, the proposed framework is unable to serialise and process the other categories of serialised RDF data. The inability of the framework is due to the limitation of Jena reasoning subsystem and the low expressivity to support semantic rules based on other categories of serialised RDF data formats. The future development on the semantic reasoning engine will consider a hybrid reasoning system that includes the processing of other non-XML based RDF data such as Entity Notation, *JSON_LD* and HDT for better-optimised performance.

References

Abburu, S. (2012). A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, *57*(17).

Abdulhafiz, W. A. & Khamis, A. (2013). Handling Data Uncertainty and Inconsistency Using Multisensor Data Fusion. *Advances in Artificial Intelligence, 2013*, 1–11.

Adelman, S., Moss, L. & Abai, M. (2005). *Data strategy* (1st edition). Addison-Wesley Professional.

Ali, S., Khusro, S., Ullah, I., Khan, A. & Khan, I. (2017). Smartontosensor: Ontology for semantic interpretation of smartphone sensors data for context-aware applications. *Journal of Sensors, 2017*.

Almeida, A. & López-de-Ipiña, D. (2011). Modelling and managing ambiguous context in intelligent environments. *V International Symposium of Ubiquitous Computing and Ambient Intelligence, UCAmI. Mexico*.

Ang, L. M., Seng, K. P., Zungeru, A. & Ijemaru, G. (2017). Big Sensor Data Systems for Smart Cities. *IEEE Internet of Things Journal, PP*(99), 1–13.

Anicic, D., Fodor, P., Rudolph, S. & Stojanovic, N. (2011). Ep-sparql: A unified language for event processing and stream reasoning. *Proceedings of the 20th international conference on World wide web*, 635–644.

Anja, W. L. K. (2009). Representing Data Quality in Sensor Data Streaming Environments. *Journal of Data and Information Quality*, *1*(2), 1–28.

Appice, A., Ciampi, A., Malerba, D. & Guccione, P. (2013). Using trend clusters for spatiotemporal interpolation of missing data in a sensor network. *Journal of Spatial Information Science, 2013*(6), 119–153.

Aufaure, M. A., Chiky, R., Curé, O., Khrouf, H. & Kepeklian, G. (2016). From Business Intelligence to semantic data stream management. *Future Generation Computer Systems, 63*, 100–107.

Avancha, S., Patel, C. & Joshi, A. (2004). Ontology-driven adaptive sensor networks. *UMBC Student Collection*.

Balakrishna, S., Thirumaran, M., Solanki, V. K. & Núñez-Valdez, E. R. (2020). Incremental hierarchical clustering driven automatic annotations for unifying iot streaming data. *International Journal of Interactive Multimedia & Artificial Intelligence, 6*(2).

Bamgboye, O., Liu, X. & Cruickshank, P. (2018). Towards Modelling and Reasoning About Uncertain Data of Sensor Measurements for Decision Support in Smart Spaces. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 744–749.

Bamgboye, O., Liu, X. & Cruickshank, P. (2019). Semantic Stream Management Framework for Data Consistency in Smart Spaces. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2*, 85–90.

Barbieri, D., Braga, D., Ceri, S., Della Valle, E., Huang, Y., Tresp, V., Rettinger, A. & Wermser, H. (2010a). Deductive and inductive stream reasoning for semantic social media analytics. *IEEE Intelligent Systems, 25*(6), 32–41.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E. & Grossniklaus, M. (2009). Continuous queries and real-time analysis of social semantic data with C-SPARQL. *CEUR Workshop Proceedings, 520*, 1–12.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E. & Grossniklaus, M. (2010b). Incremental reasoning on streams and rich background knowledge. *Extended Semantic Web Conference*, 1–15.

Barbieri, D. F., Grossniklaus, M. & Dipartimento, M. (2010c). An Execution Environment for C-SPARQL Queries. *Proceedings of the 13th International Conference on Extending Database Technology, Lausanne, Switzerland*, 441–452.

Barbieri, D. F. & Milano, P. (2014). Querying RDF Streams with C-SPARQL. *ACM Transactions on Autonomous and Adaptive Systems (TAAS) - Special section on formal methods in pervasive computing, pervasive adaptation, and self-adaptive systems: Models and algorithm*, *7*(1).

Barnaghi, P., Bermudez-Edo, M. & Tönjes, R. (2015). Challenges for Quality of Data in Smart Cities. *Journal of Data and Information Quality*, *6*(2-3), 1–4.

Beckett, D. & McBride, B. (2004). Rdf/xml syntax specification (revised). *W3C recommendation*, *10*(2.3).

Bermudez, L., Delory, E., O'Reilly, T. & del Rio Fernandez, J. (2009). Ocean observing systems demystified. *OCEANS 2009*, 1–7.

Bermudez-Edo, M., Elsaleh, T., Barnaghi, P. & Taylor, K. (2016). Iot-lite: A lightweight semantic model for the internet of things. *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, 90–97.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The semantic web. *Scientific american*, *284*(5), 34–43.

Bettini, C., Dyreson, C. E., Evans, W. S., Snodgrass, R. T. & Wang, X. S. (1998). A glossary of time granularity concepts. *Temporal databases: Research and practice* (pp. 406–413). Springer.

Bi, Z., Xu, L. D. & Wang, C. (2014). Internet of things for enterprise systems of modern manufacturing. *IEEE Transactions on Industrial Informatics*, *10*(2), 1537–1546.

Bißmeyer, N., Mauthofer, S., Bayarou, K. M. & Kargl, F. (2012). Assessment of node trustworthiness in vanets using data plausibility checks with particle filters. *2012 IEEE Vehicular Networking Conference (VNC)*, 78–85.

Blin, G., Cur 'e, O. & Faye, D. C. (2012). A survey of rdf storage approaches. *African Journal of Research in Computer Science and Applied Math*, *15*.

Bolles, A., Grawunder, M. & Jacobi, J. (2008). Streaming sparql-extending sparql to process data streams. *European Semantic Web Conference*, 448–462.

Brown, P. E., Dasu, T., Kanza, Y. & Srivastava, D. (2019). From rocks to pebbles: Smoothing spatiotemporal data streams in an overlay of sensors. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, *5*(3), 1–38.

Calbimonte, J.-P., Corcho, O. & Gray, A. J. (2010). Enabling ontology-based access to streaming data sources. *International Semantic Web Conference*, 96–111.

Calder, M., Morris, R. A. & Peri, F. (2010). Machine reasoning about anomalous sensor data. *Ecological Informatics*, *5*(1), 9–18.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. & Wilkinson, K. (2004). Jena: Implementing the semantic web recommendations. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 74–83.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S. R., Reiss, F. & Shah, M. A. (2003). Telegraphcq: Continuous dataflow processing. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 668–668.

Chao, Y., Yao, S., Buttner, W. J. & Stetter, J. R. (2005). Amperometric sensor for selective and stable hydrogen measurement. *Sensors and Actuators B: Chemical*, *106*(2), 784–790.

Cheng, R. (2003). Managing Uncertainty in Sensor Databases. *32*(4), 41–46.

Cheng, R., Kalashnikov, D. V. & Prabhakar, S. (2003). Evaluating probabilistic queries over imprecise data. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 551–562.

Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., … Taylor, K. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, *17*, 25–32.

Compton, M., Neuhaus, H., Taylor, K. & Tran, K. N. (2009). Reasoning about sensors and compositions. *CEUR Workshop Proceedings*, *522*(December), 33–48.

Corella, M. A. & Castells, P. (2006). Semi-automatic semantic-based Web service classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4103 LNCS*, 459–470.

D'Aniello, G., Gaeta, M., Orciuoli, F. & Simonetti, A. (2018). An approach based on semantic stream reasoning to support decision processes in smart cities. *Telematics and Informatics*, *35*(6), 1795.

Dell Aglio, D., Della Valle, E., van Harmelen, F. & Bernstein, A. (2017). Stream reasoning: A survey and outlook. *Data Science*, *1*(1-2), 59–83.

Dell'Aglio, D. & Della Valle, E. (2014). Incremental reasoning on rdf streams.

Dell'Aglio, D., Della Valle, E., Calbimonte, J.-P. & Corcho, O. (2014). Rsp-ql semantics: A unifying query model to explain heterogeneity of rdf stream processing systems. *International Journal on Semantic Web and Information Systems (IJSWIS)*, *10*(4), 17–44.

Dhillon, S. S., Chakrabarty, K. & Iyengar, S. S. (2002). Sensor placement for grid coverage under imprecise detections. *Proceedings of the Fifth International Conference on Information Fusion.*, *2*, 1581–1587.

Dividino, R., Soares, A., Matwin, S., Isenor, A. W., Webb, S. & Brousseau, M. (2018). *Semantic integration of real-time heterogeneous data streams for ocean-related decision making*. Defence Research; Development Canada.

Eid, M., Liscano, R. & El Saddik, A. (2007). A universal ontology for sensor networks data. *2007 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, 59–62.

Elnahrawy, E. & Nath, B. (2003). Cleaning and querying noisy sensors. *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 78.

Eppler, M. J. (2006). *Managing information quality: Increasing the value of information in knowledge-intensive products and processes*. Springer Science & Business Media.

Fensel, D. (2001). Ontologies: Dynamic Networks Formally Represented Meaning. *Proceeding of the International Semantic Web Working Symposium (SWWS)*.

Forgy, C. L. (1989). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Readings in artificial intelligence and databases* (pp. 547–559). Elsevier.

Garofalakis, M., Gehrke, J. & Rastogi, R. (2016). *Data stream management: Processing high-speed data streams*. Springer.

Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O. & Schaub, T. (2013). Answer set programming for stream reasoning. *arXiv preprint*.

Golab, L. & Özsu, M. T. (2003). Processing sliding window multi-joins in continuous queries over data streams. *Proceedings 2003 VLDB Conference*, 500–511.

Gruber, T. R. et al. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, *5*(2), 199–221.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, *43*(5-6), 907–928.

Gu, T., Wang, X. H., Pung, H. K. & Zhang, D. Q. (2004). An Ontology-based Context Model in Intelligent Environments. *in Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 270–275.

Gyrard, A. (2013). A Machine-to-machine Architecture to Merge Semantic Sensor Measurements. *Proceedings of the 22Nd International Conference on World Wide Web Companion*, 371–376.

Gyrard, A. & Serrano, M. (2016). A Unified Semantic Engine for Internet of Things and Smart Cities: From Sensor Data to End-Users Applications. *Proceedings - 2015 IEEE International Conference on Data Science and Data Intensive Systems*, 718–725.

Haller, A., Janowicz, K., Cox, S. J., Lefrançois, M., Taylor, K., Le Phuoc, D., Lieberman, J., Garcıa-Castro, R., Atkinson, R. & Stadler, C. (2018). The sosa/ssn ontology: A joint wec and ogc standard specifying the semantics of sensors observations actuation and sampling. *Semantic web* (pp. 1–19). IOS Press.

Hammad, M. A., Mokbel, M. F., Ali, M. H., Aref, W. G., Catlin, A. C., Elmagarmid, A. K., Eltabakh, M., Elfeky, M. G., Ghanem, T. M., Gwadera, R. et al. (2004). Nile: A query processing engine for data streams. *Proceedings. 20th International Conference on Data Engineering*, 851.

Hoeksema, J. & Kotoulas, S. (2011). High-performance distributed stream reasoning using s4. *Ordring Workshop at ISWC*.

Holger Neuhaus, M. C. (2009). The Semantic Sensor Network: Ontology A Generic Language to Describe Sensor Assets. *Proceedings of the AGILE 2009 Pre-Conference Workshop Challenges in Geospatial Data Harmonisation, Hannover, Germany*.

Horridge, M. (n.d.). Owl syntaxes [online]. january 22, 2010. *Internet Available: http://ontogenesis. knowledgeblog. org/88 (22 January, 2010)(Accessed: January 25, 2020)*.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M. et al. (2004). Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, *21*(79), 1–31.

Jajaga, E. & Ahmedi, L. (2017). C-SWRL: A Unique Semantic Web Framework for Reasoning Over Stream Data. *International Journal of Semantic Computing*, *11*(03), 391–409.

Jajaga, E., Ahmedi, L. & Ahmedi, F. (2015). An expert system for water quality monitoring based on ontology. *Research Conference on Metadata and Semantics Research*, 89–100.

Janowicz, K., Haller, A., Cox, S. J., Le Phuoc, D. & Lefrançois, M. (2019). Sosa: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics*, *56*, 1–10.

Jekjantuk, N., Pan, J. Z. & Alexopoulos, P. (2016). Towards a Meta-Reasoning Framework for Reasoning about Vagueness in OWL Ontologies. *Proceedings - 2016 IEEE 10th International Conference on Semantic Computing, ICSC 2016*, 222–229.

Kaelbling, L. P. et al. (1987). An architecture for intelligent reactive systems. *Reasoning about actions and plans*, 395–410.

Kamilaris, A., Gao, F., Prenafeta-Boldu, F. X. & Ali, M. I. (2016). Agri-iot: A semantic framework for internet of things-enabled smart farming applications. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 442–447.

Karkouch, A., Mousannif, H., Al Moatassime, H. & Noel, T. (2016). Data quality in internet of things: A state-of-the-art survey. *Journal of Network and Computer Applications*, *73*, 57–81.

Kehoe, M., Cosgrove, M., Gennaro, S., Harrison, C., Harthoorn, W., Hogan, J., Meegan, J., Nesbitt, P. & Peters, C. (2011). Smarter cities series: A foundation for understanding ibm smarter cities. *Redguides for Business Leaders, IBM*, REDP–4733.

Kim, J.-h., Kim, J.-h., Kwon, H., Kim, D.-h., Kwak, H.-y. & Lee, S.-j. (2008). Building a Service-Oriented Ontology for Wireless Sensor Networks Building a Service-Oriented Ontology for Wireless Sensor Networks. *7th IEEE/ACIS International Conference on Computer and Information Science , Portland, Ore, USA*, 649–654.

Kläs, M. & Vollmer, A. M. (2018). Uncertainty in machine learning applications: A practice-driven classification of uncertainty. *International Conference on Computer Safety, Reliability, and Security*, 431–438.

Komazec, S. & Cerri, D. (2012). Sparkwave : Continuous schema-enhanced pattern matching over RDF data Streams. *In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, 58–68.

Korpipää, P. & Mäntyjärvi, J. (2003). An ontology for mobile device sensor-based context awareness. *International and Interdisciplinary Conference on Modeling and Using Context*, 451–458.

Krämer, J. & Seeger, B. (2004). *A temporal foundation for continuous queries over data streams*. Univ.

Kuemper, D., Iggena, T., Fischer, M., Toenjes, R. & Pulvermueller, E. (2016). Monitoring data stream reliability in smart city environments. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 571–576.

Kumar, M., Garg, D. & Zachery, R. (2006). A generalized approach for inconsistency detection in data fusion from multiple sensors. *2006 American Control Conference*, (July 2006), 2078–2083.

Kumar, M. (2015). Building agile data driven smart cities. *IDC White paper*.

Lehmann, J. & Völker, J. (2014). *Perspectives on ontology learning* (Vol. 18). IOS Press.

Le-Phuoc, D., Dao-Tran, M., Parreira, J. X. & Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data. *International Semantic Web Conference*, 370–388.

Leyk, S., Boesch, R. & Weibel, R. (2005). A conceptual framework for uncertainty investigation in map-based land cover change modelling. *Transactions in GIS*, *9*(3), 291–322.

Li, L., Peng, T. & Kennedy, J. (2011). A rule based taxonomy of dirty data. *GSTF Journal on Computing*, *1*(2).

Liu, H., Wu, Y. & Yang, Y. (2017). Analogical inference for multi-relational embeddings. *arXiv preprint arXiv:1705.02426*.

Maarala, A. I., Su, X. & Riekki, J. (2017). Semantic Reasoning for Context-Aware Internet of Things Applications. *IEEE Internet of Things Journal*, *4*(2), 461–473.

Margara, A., Urbani, J., Van Harmelen, F. & Bal, H. (2014). Streaming the Web: Reasoning over dynamic data. *Journal of Web Semantics*, *25*, 24–44.

Mishra, N., Lin, C.-C. & Chang, H.-T. (2015). A cognitive adopted framework for iot big-data management and knowledge discovery prospective. *International Journal of Distributed Sensor Networks*, *11*(10), 718390.

Nickel, M., Tresp, V. & Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. *Icml*, *11*, 809–816.

Paulus, D., de Vries, G. & Van de Walle, B. (2019). Effects of data ambiguity and cognitive biases on the interpretability of machine learning models in humanitarian decision making. *arXiv preprint*.

Pease, A., Niles, I. & Li, J. (2002). The suggested upper merged ontology: A large ontology for the semantic web and its applications. *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*, *28*, 7–10.

Pham, T.-l., Ali, M. I. & Mileo, A. (2019). C-ASP : Continuous ASP-Based Reasoning over RDF Streams C-ASP : Continuous ASP-based Reasoning over RDF Streams. *In*

International Conference on Logic Programming and Nonmonotonic Reasoning Springer., (January), 45–50.

Pipino, L. L., Lee, Y. W. & Wang, R. Y. (2002). Data quality assessment. *Communications of the ACM*, *45*(4), 211.

Poli, R., Healy, M. & Kameas, A. (2010). *Theory and applications of ontology: Computer applications*. Springer.

Powers, D. M. (2020). Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

Prabhakar, S. (n.d.). Orion: Managing uncertain (sensor) data.

Psyllidis, A. (2015). Ontology-Based Data Integration from Heterogeneous Urban Systems : A Knowledge Representation Framework for Smart Cities. *The 14th International Conference on Computers in Urban Planning and Urban Management*.

Rao, A. P. (2019). Quality measures for semantic web application. *Web services: Concepts, methodologies, tools, and applications* (pp. 1907–1916). IGI Global.

Ren, Y., Pan, J. Z. & Zhao, Y. (2010). Towards scalable reasoning on ontology streams via syntactic approximation. *Proc. of IWOD*.

Rinne, M., Abdullah, H., Törmä, S. & Nuutila, E. (2012a). Processing heterogeneous rdf events with standing sparql update rules. *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, 797–806.

Rinne, M., Nuutila, E. & Törmä, S. (2012b). Instans: High-performance event processing with standard rdf and sparql. *Proceedings of the 2012th International Conference on Posters & Demonstrations Track*, *914*, 101–104.

Roberge, B. (2000). Effect of varying the combustion parameters on the emissions of carbon monoxide and nitrogen oxides in the exhaust gases from propane-fueled vehicles. *Applied occupational and environmental hygiene*, *15*(5), 421–428.

Russomanno, D. J., Kothari, C. R. & Thomas, O. A. (2005). Building a Sensor Ontology : A Practical Approach Leveraging ISO and OGC Models. *International Conference on Artificial Intelligence*, 637–643.

Sandra Geisler, Sven Weber & Quix, C. (2011). Ontology-Based Data Quality Framework for Data Stream Applications. *Proc. of the 16th International Conference on Information Quality, Adelaide, AUS*, *9*, 115–129.

Sta, H. B. (2016). Quality and the efficiency of data in "Smart-Cities". *Future Generation Computer Systems*.

Stojanovic, L. (2004). Methods and tools for ontology evolution. *University of Karlsruhe*.

Stonebraker, M., Çetintemel, U. & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM Sigmod Record*, *34*(4), 42–47.

Stuckenschmidt, H., Ceri, S., Della Valle, E. & Van Harmelen, F. (2010). Towards expressive stream reasoning. *Dagstuhl Seminar Proceedings*.

Su, X., Riekki, J., Nurminen, J. K., Nieminen, J. & Koskimies, M. (2015). Adding semantics to internet of things. *Concurrency and Computation: Practice and Experience*, *27*(8), 1844–1860.

Suárez-Figueroa, M. C., Gómez-Pérez, A. & Fernández-López, M. (2012). The neon methodology for ontology engineering. *Ontology engineering in a networked world* (pp. 9–34). Springer.

Tallevi-Diotallevi, S., Kotoulas, S., Foschini, L., Lécué, F. & Corradi, A. (2013). Real-time urban monitoring in Dublin using semantic and stream technologies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8219 LNCS*(PART 2), 178–194.

Tan, Y., Sehgal, V. & Shahri, H. (2005). Sensoclean: Handling noisy and incomplete data in sensor networks using modeling. *Main*, 1–18.

Thomas, E., Pan, J. Z. & Ren, Y. (2010). Trowl: Tractable owl 2 reasoning infrastructure. *Extended Semantic Web Conference*, 431–435.

Tu, D. Q., Kayes, A., Rahayu, W. & Nguyen, K. (2020). Iot streaming data integration from multiple sources. *Computing*, *102*(10), 2299–2329.

Urbani, J., Margara, A., Jacobs, C., Van Harmelen, F. & Bal, H. (2013). Dynamite: Parallel materialization of dynamic rdf data. *International Semantic Web Conference*, 657–672.

Valle, E. D., Ceri, S., Harmelen, F. v. & Fensel, D. (2009). It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems, 24*(6), 83–89.

Vongsingthong, S. & Smanchat, S. (2015). A review of data management in internet of things. *Asia-Pacific Journal of Science and Technology, 20*(2), 215–240.

Wang, M., Zhang, G., Zhang, C., Zhang, J. & Li, C. (2013). An IoT-based appliance control system for smart homes. *Proceedings of the 2013 International Conference on Intelligent Control and Information Processing*, 744–747.

Wang, Q., Mao, Z., Wang, B. & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering, 29*(12), 2724–2743.

Wang, X., Dong, J. S., Chin, C., Hettiarachchi, S. R. & Zhang, D. (2004). Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing, 3*(3), 32–39.

Wang, Z., Zhang, J., Feng, J. & Chen, Z. (2014). Knowledge graph and text jointly embedding. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1591–1601.

Yoon, J., Zame, W. R. & van der Schaar, M. (2018). Estimating missing data in temporal data streams using multi-directional recurrent neural networks. *IEEE Transactions on Biomedical Engineering, 66*(5), 1477–1490.

Zhang, Y., Duc, P. M., Corcho, O. & Calbimonte, J.-P. (2012). Srbench: A streaming rdf/sparql benchmark. *International Semantic Web Conference*, 641–657.

Zhou, Q. & Fikes, R. (2002). A reusable time ontology. *Proceeding of the AAAI Workshop on Ontologies for the Semantic Web*.

Appendix A

Glossary

A.1 Glossary

- **ActiveMQ :** It is an Apache Message Oriented Middleware (MOM) which is often implemented with Java Mssage Service(JMS) specification.
- **Ambient temperature :** It describes the temperature of the air around an object indoor or outdoor
- **Event :** An event is anything that happens within a specified time and that can be captured within a system or an enterprise.
- **Materialization :**This corresponds to the technique of computation and storage of inferred triples to enhance query performance .
- **Stream :** A stream is a continuous flow of event object that are produced by numerous connected devices, IoT and other types of sensors.
- **W3C :** An international Organisation that promotes standard for web and interoperability between web products through provision of specifications and reference software

B.1 Prototype Application Screenshots

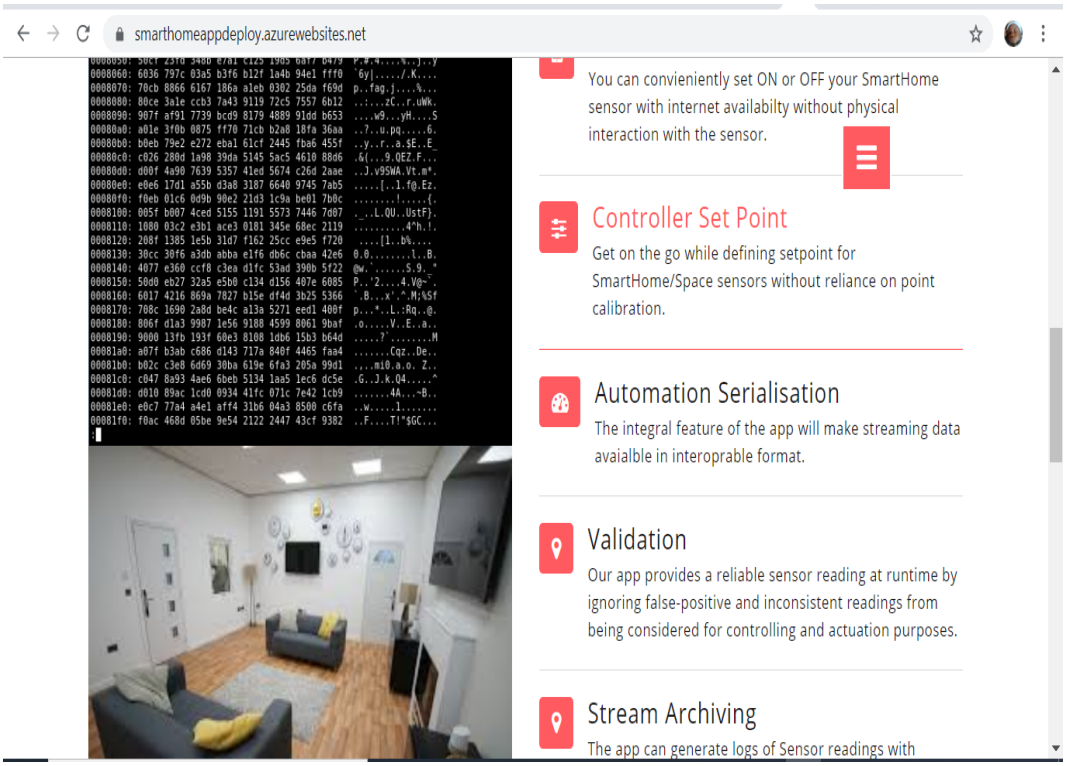

Figure B.1: Summary page of web Interface

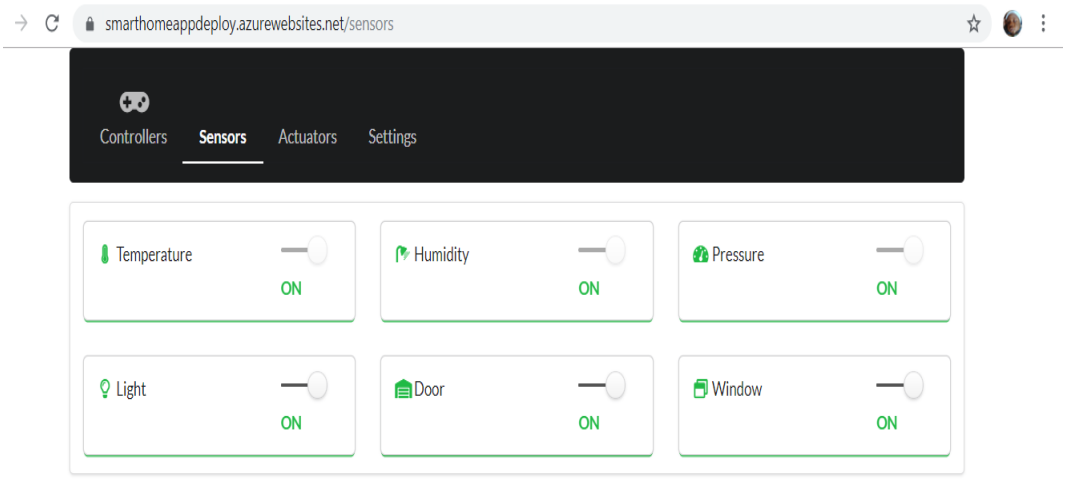

Figure B.2: Sensor State Screenshot

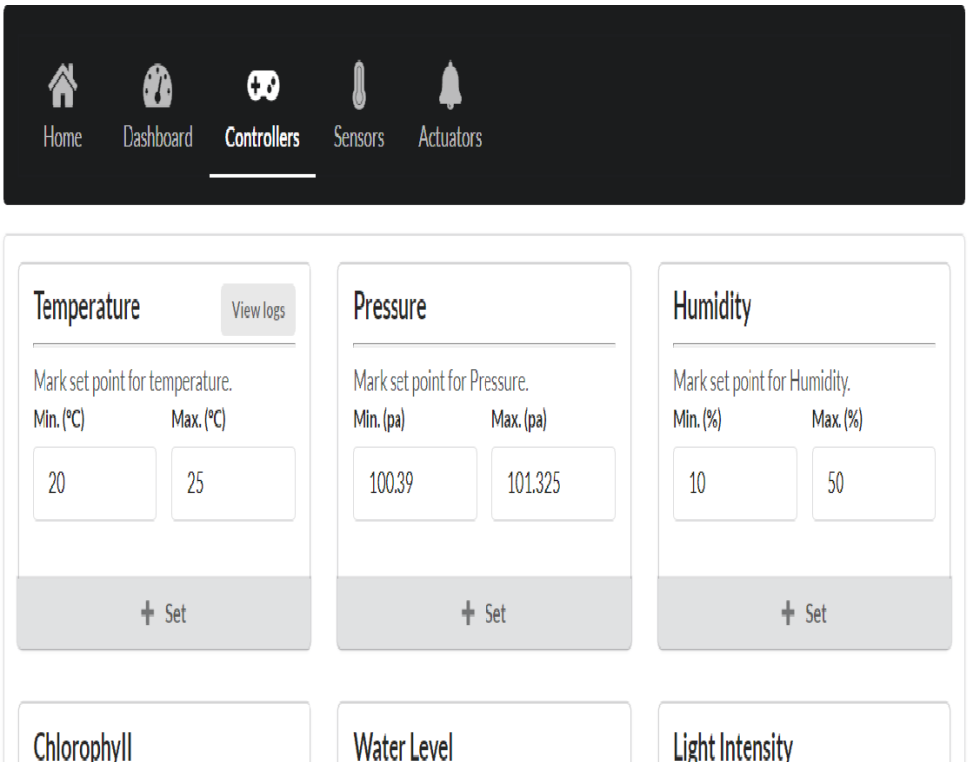

Figure B.3: Controller Set-Point Screenshot

Appendix C

Sample Air Quality Data Set

Date	Time	CO(GT)	NMHC(GT)	C6H6(GT)	Nox
10/03/2004	18:00:00	2.6	150	11.9	166
10/03/2004	19:00:00	2	112	9.4	103
10/03/2004	20:00:00	2.2	88	9	131
10/03/2004	21:00:00	2.2	80	9.2	172
10/03/2004	22:00:00	1.6	51	6.5	131
11/03/2004	16:00:00	2.2	95	8.3	193
11/03/2004	17:00:00	2.9	150	11.2	243
11/03/2004	18:00:00	4.8	307	20.8	281
11/03/2004	19:00:00	6.9	461	27.4	383
11/03/2004	20:00:00	6.1	401	24	351
11/03/2004	21:00:00	3.9	197	12.8	240
11/03/2004	22:00:00	1.5	61	4.7	94
11/03/2004	23:00:00	1	26	2.6	47
12/03/2004	00:00:00	1.7	55	5.9	122
12/03/2004	01:00:00	1.9	53	6.4	133

12/03/2004	02:00:00	1.4	40	4.1	82
12/03/2004	03:00:00	0.8	21	1.9	-200
12/03/2004	04:00:00	-200	10	1.1	21
12/03/2004	05:00:00	0.6	7	1	30
12/03/2004	06:00:00	0.8	17	1.8	56
12/03/2004	07:00:00	1.4	33	4.4	109
12/03/2004	08:00:00	4.4	202	17.9	307
12/03/2004	09:00:00	-200	-200	22.1	-200
12/03/2004	10:00:00	3.1	208	14	187
12/03/2004	11:00:00	2.7	166	11.6	216
12/03/2004	12:00:00	2.1	114	10.2	143
12/03/2004	13:00:00	2.5	140	11	160
12/03/2004	14:00:00	2.7	169	12.8	163
12/03/2004	15:00:00	2.9	185	14.2	190
12/03/2004	16:00:00	2.8	165	12.7	178
12/03/2004	17:00:00	2.4	133	11.7	150
12/03/2004	18:00:00	3.9	233	19.3	206
12/03/2004	19:00:00	3.7	242	18.2	202
12/03/2004	20:00:00	6.6	488	32.6	340
12/03/2004	21:00:00	4.4	333	20.1	274
12/03/2004	22:00:00	3.5	215	14.3	253
12/03/2004	23:00:00	5.4	367	21.8	300
13/03/2004	00:00:00	2.7	122	9.6	193

13/03/2004	01:00:00	1.9	67	7.4	139
13/03/2004	02:00:00	1.6	43	5.4	83
13/03/2004	03:00:00	1.7	46	5.4	-200
13/03/2004	04:00:00	-200	56	6.2	109
13/03/2004	05:00:00	1	30	2.6	62
13/03/2004	06:00:00	1.2	27	2.9	53
13/03/2004	07:00:00	1.5	47	5.1	139
13/03/2004	08:00:00	2.7	132	11.8	256
13/03/2004	09:00:00	3.7	239	15.1	295
13/03/2004	10:00:00	3.2	160	12.9	250
13/03/2004	11:00:00	4.1	283	16.1	296
13/03/2004	12:00:00	3.6	210	14	239
13/03/2004	13:00:00	2.8	154	12.3	153
13/03/2004	14:00:00	2	112	8.6	118
13/03/2004	15:00:00	2	108	9.2	119
13/03/2004	16:00:00	2.5	111	10.2	138
13/03/2004	17:00:00	2.3	97	10.6	148
13/03/2004	18:00:00	3.2	191	15.5	227
13/03/2004	19:00:00	4.2	258	19.6	277
13/03/2004	20:00:00	4.2	284	19.2	279
13/03/2004	21:00:00	4.2	269	18.3	283
13/03/2004	22:00:00	3.1	180	13.1	214
13/03/2004	23:00:00	2.6	116	10.9	172

14/03/2004	00:00:00	2.9	93	11	190
14/03/2004	01:00:00	2.8	131	11.9	174
14/03/2004	02:00:00	2.5	92	8.6	128
14/03/2004	03:00:00	2.4	132	9.7	-200
14/03/2004	04:00:00	-200	56	5.2	70
14/03/2004	05:00:00	1.2	32	3.7	53
14/03/2004	06:00:00	1	29	2.5	44
14/03/2004	07:00:00	0.9	27	2.4	74
14/03/2004	08:00:00	1.4	36	4.2	101
14/03/2004	09:00:00	1.6	57	6.4	118
14/03/2004	10:00:00	2.2	129	8.6	144
14/03/2004	11:00:00	2.8	148	10.9	176
14/03/2004	12:00:00	2.8	145	10.7	161
15/03/2004	02:00:00	1.8	66	7	108
15/03/2004	03:00:00	1.1	44	4.4	-200
15/03/2004	04:00:00	-200	44	4	66
15/03/2004	05:00:00	1	39	3.9	88
15/03/2004	06:00:00	1.4	51	6.4	138
15/03/2004	07:00:00	2.2	107	9.7	228
15/03/2004	08:00:00	5.5	336	25.9	360
15/03/2004	09:00:00	8.1	618	36.7	478
15/03/2004	10:00:00	5.8	438	26.6	394
15/03/2004	11:00:00	4.2	334	20.1	319

15/03/2004	12:00:00	3.1	221	14.1	201
15/03/2004	13:00:00	2.9	207	14.9	171
15/03/2004	14:00:00	2.9	191	15.4	159
15/03/2004	15:00:00	2.5	185	12.1	153
15/03/2004	16:00:00	2.3	141	11.5	143
15/03/2004	17:00:00	2.8	214	14.8	156
15/03/2004	18:00:00	6.1	471	32.1	314
16/03/2004	15:00:00	2.8	228	14.6	180
16/03/2004	16:00:00	2.9	201	16.6	184
16/03/2004	17:00:00	2.9	199	15.8	190
16/03/2004	18:00:00	3.4	237	17.8	184
16/03/2004	19:00:00	3.9	261	19.1	181
16/03/2004	20:00:00	3.2	230	15.8	166
16/03/2004	21:00:00	5.1	349	24.9	317
16/03/2004	22:00:00	2.6	183	13.5	184
16/03/2004	23:00:00	1.7	88	9.1	130
17/03/2004	00:00:00	1.7	85	8.6	132
17/03/2004	01:00:00	1.2	47	5.4	95
17/03/2004	02:00:00	0.9	34	4.1	70
17/03/2004	03:00:00	0.7	26	2.6	-200
17/03/2004	04:00:00	-200	17	1.9	54
17/03/2004	05:00:00	0.5	11	1.6	28
18/03/2004	04:00:00	-200	28	2.9	60

18/03/2004	05:00:00	0.6	20	2.5	37
18/03/2004	06:00:00	0.7	26	3	68
18/03/2004	07:00:00	1.5	78	7.7	139
18/03/2004	08:00:00	4.7	319	23.3	339
18/03/2004	09:00:00	6.6	506	35.8	421
18/03/2004	10:00:00	4.5	-200	21.3	349
18/03/2004	11:00:00	2.8	-200	14.3	224
18/03/2004	12:00:00	2.2	-200	12.5	171
18/03/2004	13:00:00	2.2	-200	12.2	149
18/03/2004	14:00:00	2.3	-200	13.1	137
18/03/2004	15:00:00	2.2	-200	14.4	149
18/03/2004	16:00:00	2.8	-200	16.8	172
18/03/2004	17:00:00	2.7	-200	14.5	166
18/03/2004	18:00:00	3.7	-200	21.5	214
18/03/2004	19:00:00	5.1	-200	26.4	280
18/03/2004	20:00:00	5.1	-200	26	276
18/03/2004	21:00:00	3.2	-200	14.1	178
18/03/2004	22:00:00	2.1	-200	10.3	129
18/03/2004	23:00:00	1.7	-200	8.3	95
19/03/2004	00:00:00	2	-200	8.9	126
19/03/2004	01:00:00	1.6	-200	6.6	103
19/03/2004	02:00:00	0.9	-200	3.6	48
19/03/2004	03:00:00	0.7	-200	2.5	-200

19/03/2004	04:00:00	-200	-200	1.8	20
19/03/2004	05:00:00	0.5	-200	1.3	18
19/03/2004	06:00:00	0.7	-200	2.3	56
19/03/2004	07:00:00	1.5	-200	6.7	115
19/03/2004	08:00:00	4.8	-200	22.8	320
19/03/2004	09:00:00	6.2	-200	31.3	357
19/03/2004	10:00:00	4	-200	19.2	253
19/03/2004	11:00:00	3.3	-200	16.4	218
19/03/2004	12:00:00	2.8	-200	14	192
19/03/2004	13:00:00	3	-200	15.3	176
19/03/2004	14:00:00	3.3	-200	16.7	198
20/03/2004	09:00:00	2.4	-200	10.5	147
20/03/2004	10:00:00	2.6	-200	12.5	166
20/03/2004	11:00:00	2.8	-200	12.3	195
20/03/2004	12:00:00	2.6	-200	11.7	182
20/03/2004	13:00:00	2.6	-200	11.7	168
20/03/2004	14:00:00	2.1	-200	9.3	125
20/03/2004	15:00:00	1.7	-200	7.6	95
20/03/2004	16:00:00	1.6	-200	6.7	79
20/03/2004	17:00:00	2.1	-200	9.7	119
20/03/2004	18:00:00	2.3	-200	12.4	142
20/03/2004	19:00:00	3.5	-200	16.6	215
20/03/2004	20:00:00	3.9	-200	16.4	229

20/03/2004	21:00:00	3.3	-200	13.7	206
20/03/2004	22:00:00	2.3	-200	9.9	147
20/03/2004	23:00:00	2.1	-200	8.9	122
21/03/2004	00:00:00	2.8	-200	10.6	175
21/03/2004	01:00:00	2.1	-200	7.4	133
21/03/2004	02:00:00	1.6	-200	6.2	89
21/03/2004	03:00:00	1.6	-200	6.5	-200
21/03/2004	04:00:00	-200	-200	5.8	85
21/03/2004	18:00:00	3.8	-200	15.1	173
21/03/2004	19:00:00	3.5	-200	12.6	185
21/03/2004	20:00:00	4.3	-200	15.1	266
21/03/2004	21:00:00	2.8	-200	9.9	188
21/03/2004	22:00:00	1.9	-200	8	122
21/03/2004	23:00:00	1.9	-200	7.9	112
22/03/2004	00:00:00	1.7	-200	6.1	93
22/03/2004	01:00:00	1.5	-200	5.1	74
22/03/2004	02:00:00	0.6	-200	1.7	23
22/03/2004	03:00:00	0.4	-200	0.7	-200
22/03/2004	04:00:00	-200	-200	0.8	17
22/03/2004	05:00:00	0.3	-200	0.7	15
22/03/2004	06:00:00	0.6	-200	2	41
22/03/2004	07:00:00	1.2	-200	5.1	86
22/03/2004	08:00:00	3.6	-200	17.7	226

22/03/2004	09:00:00	3.7	-200	18.4	214
22/03/2004	10:00:00	1.8	-200	6.9	115
22/03/2004	11:00:00	1.6	-200	7.3	124
22/03/2004	12:00:00	1.9	-200	9.6	122
25/03/2004	03:00:00	0.7	35	1.4	-200
25/03/2004	04:00:00	0.5	29	0.9	18
25/03/2004	05:00:00	0.5	21	0.6	12
25/03/2004	06:00:00	0.6	46	1.4	43
25/03/2004	07:00:00	1.1	55	3.8	84
25/03/2004	08:00:00	2.7	271	11.6	184
25/03/2004	09:00:00	3.5	434	17.8	202
25/03/2004	10:00:00	2.3	300	8.8	133
25/03/2004	11:00:00	1.6	116	6.8	130
25/03/2004	12:00:00	1.3	95	5.9	106
25/03/2004	13:00:00	2	211	8.9	132
25/03/2004	14:00:00	1.9	168	8.2	106
25/03/2004	15:00:00	1.9	154	8.6	125
25/03/2004	16:00:00	2.2	267	10.1	138
25/03/2004	17:00:00	2	143	9.4	120
25/03/2004	18:00:00	2.9	374	14.6	158
25/03/2004	19:00:00	5.2	797	24.6	253
25/03/2004	20:00:00	4.6	698	21.6	231
25/03/2004	21:00:00	2.5	234	10.3	150

25/03/2004	22:00:00	1.5	104	5.7	99
25/03/2004	23:00:00	1.2	67	4.5	75
26/03/2004	00:00:00	1.7	88	5.5	93
26/03/2004	01:00:00	1.4	79	4.8	79
26/03/2004	02:00:00	1.2	61	3.6	67
26/03/2004	03:00:00	0.6	43	1.7	-200
26/03/2004	04:00:00	0.7	40	2.2	45
26/03/2004	05:00:00	0.8	52	3	72
26/03/2004	06:00:00	0.9	64	4	103
26/03/2004	07:00:00	1.6	88	6.7	132
26/03/2004	08:00:00	3.4	375	16.7	239
26/03/2004	09:00:00	3.8	592	19.3	275
26/03/2004	10:00:00	3.1	357	14.8	232
26/03/2004	11:00:00	2.7	296	13.4	180
26/03/2004	12:00:00	2	181	11	112
26/03/2004	13:00:00	2.3	211	12.5	116
26/03/2004	14:00:00	1.9	199	8.4	103
26/03/2004	15:00:00	1.3	81	5.3	89
26/03/2004	16:00:00	1.9	143	8.8	112
26/03/2004	17:00:00	2.3	247	11.2	127
26/03/2004	18:00:00	2.4	239	11.6	125
26/03/2004	19:00:00	2.7	267	12.4	120
26/03/2004	20:00:00	2.6	261	10.6	120

26/03/2004	21:00:00	1.5	97	6	99
26/03/2004	22:00:00	1.2	66	4.6	79
26/03/2004	23:00:00	1.1	60	4.1	66
27/03/2004	00:00:00	1.5	77	5.2	94
27/03/2004	01:00:00	1	57	3.2	70
27/03/2004	02:00:00	1.2	65	4.5	75
27/03/2004	03:00:00	1.1	59	3.8	-200
27/03/2004	04:00:00	-200	48	3.8	57
27/03/2004	05:00:00	0.8	27	1.9	32
27/03/2004	06:00:00	0.9	25	2.4	54
27/03/2004	07:00:00	1.1	42	3.3	68
27/03/2004	08:00:00	1.5	78	6.7	98
27/03/2004	09:00:00	1.8	128	8.5	128
27/03/2004	10:00:00	2.1	184	9.7	129
27/03/2004	11:00:00	2.1	156	9.4	130
27/03/2004	12:00:00	1.9	176	9	111
27/03/2004	13:00:00	2.1	232	10	106
27/03/2004	14:00:00	2.5	305	12.6	137
27/03/2004	15:00:00	1.9	150	7.6	113
27/03/2004	16:00:00	2.2	188	11.8	122
27/03/2004	17:00:00	2.3	221	11.2	137
27/03/2004	18:00:00	2.7	219	12.4	165
27/03/2004	19:00:00	3	306	12.9	196

27/03/2004	20:00:00	2.8	270	12.2	174
27/03/2004	21:00:00	2.2	231	8.8	140
27/03/2004	22:00:00	1.6	125	6.8	102
27/03/2004	23:00:00	2.1	122	8.6	130
28/03/2004	00:00:00	2.3	161	8.9	121
28/03/2004	01:00:00	2.3	101	8.3	111
28/03/2004	02:00:00	1.7	95	6.3	87
28/03/2004	03:00:00	2.2	129	8.3	-200
28/03/2004	04:00:00	1.3	96	5.1	77
28/03/2004	05:00:00	0.8	54	2.6	39
28/03/2004	06:00:00	1.1	63	3.6	77
28/03/2004	07:00:00	1.4	72	3.6	91
28/03/2004	08:00:00	1.3	91	4.7	105
28/03/2004	09:00:00	1.9	127	7.5	133
28/03/2004	10:00:00	2.3	193	9	128
28/03/2004	11:00:00	2.3	188	9.5	126
28/03/2004	12:00:00	1.8	151	7.7	92
28/03/2004	13:00:00	1.4	103	5.7	66
28/03/2004	14:00:00	1	55	3.8	50
28/03/2004	15:00:00	1.4	104	4.8	70
28/03/2004	16:00:00	1.3	116	4.3	73
28/03/2004	17:00:00	1.3	93	4.1	77
28/03/2004	18:00:00	1.4	93	4.7	86

28/03/2004	19:00:00	1.9	155	6.2	93
28/03/2004	20:00:00	1.8	115	5.5	105
28/03/2004	21:00:00	1.1	75	3.3	67
28/03/2004	22:00:00	1.1	65	3.2	63
28/03/2004	23:00:00	1.1	57	2.9	63
29/03/2004	00:00:00	0.9	40	2.2	46
29/03/2004	01:00:00	0.6	27	1.3	21
29/03/2004	02:00:00	0.5	23	1.1	22
29/03/2004	03:00:00	0.7	28	1.3	-200
29/03/2004	04:00:00	0.6	21	1.2	39
29/03/2004	05:00:00	0.7	33	1.7	55
29/03/2004	06:00:00	0.9	40	2.9	76
29/03/2004	07:00:00	2.9	279	14.3	181
29/03/2004	08:00:00	4.1	743	19.7	259
29/03/2004	09:00:00	1.5	147	5.5	118
29/03/2004	10:00:00	1.5	97	5.6	119
29/03/2004	11:00:00	1.5	118	5.8	123
29/03/2004	12:00:00	1.4	91	5.5	92
29/03/2004	13:00:00	1.6	146	6.5	91
29/03/2004	14:00:00	1.5	139	5.5	103
29/03/2004	15:00:00	1.4	155	5.2	102
29/03/2004	16:00:00	1.5	128	5.8	94
29/03/2004	17:00:00	1.9	166	7.9	98

29/03/2004	18:00:00	2.5	299	10.2	126
29/03/2004	19:00:00	2.1	163	8.2	108
29/03/2004	20:00:00	1.6	154	5.7	95
29/03/2004	21:00:00	1.2	80	3.8	78
29/03/2004	22:00:00	1.1	58	4	61
29/03/2004	23:00:00	1	55	2.8	55
30/03/2004	00:00:00	1	33	2.6	57
30/03/2004	01:00:00	0.7	33	1.8	41
30/03/2004	02:00:00	0.7	32	1.7	46
30/03/2004	03:00:00	0.8	25	1.4	-200
30/03/2004	04:00:00	-200	29	1.3	41
30/03/2004	05:00:00	0.7	26	2.3	52
30/03/2004	06:00:00	1.1	86	5.3	111
30/03/2004	07:00:00	2.6	294	13.4	191
30/03/2004	08:00:00	4	664	23.8	244
30/03/2004	09:00:00	4.2	695	21.5	283
30/03/2004	10:00:00	4.7	735	21	320
30/03/2004	11:00:00	3.9	649	18.4	249
30/03/2004	12:00:00	3.7	586	18.9	219
30/03/2004	13:00:00	3.4	546	17.1	200
30/03/2004	14:00:00	2.2	245	10.4	138
30/03/2004	15:00:00	1.9	178	8	118
30/03/2004	16:00:00	1.6	130	6.2	99

30/03/2004	17:00:00	2.1	151	9.7	112
30/03/2004	18:00:00	2.2	272	9.6	117
30/03/2004	19:00:00	2.7	301	11.9	129
30/03/2004	20:00:00	2.4	237	8.7	132
30/03/2004	21:00:00	1.3	95	4.7	73
30/03/2004	22:00:00	1.2	68	3.8	68
30/03/2004	23:00:00	1.5	101	4.7	80
31/03/2004	00:00:00	1.3	81	3.8	68
31/03/2004	01:00:00	1	50	2.7	44
31/03/2004	02:00:00	0.9	66	2.8	47
31/03/2004	03:00:00	0.5	22	1	-200
31/03/2004	04:00:00	0.5	18	0.8	15
31/03/2004	05:00:00	0.6	31	1.5	44
31/03/2004	06:00:00	1	57	3.5	85
31/03/2004	07:00:00	3.1	342	15.6	207
31/03/2004	08:00:00	4.1	644	19.9	230
31/03/2004	09:00:00	2.2	216	8.6	181
31/03/2004	10:00:00	1.7	117	6.5	144
31/03/2004	11:00:00	1.9	156	7.7	140
31/03/2004	12:00:00	2.9	332	11.3	204
31/03/2004	13:00:00	2.2	232	9.1	149
01/04/2004	21:00:00	2.5	254	10.8	154
01/04/2004	22:00:00	2	188	9.9	127

01/04/2004	23:00:00	2	120	8.5	122
02/04/2004	00:00:00	2	157	8	126
02/04/2004	01:00:00	1.3	88	5.6	84
02/04/2004	02:00:00	1	68	3.2	58
02/04/2004	03:00:00	0.9	57	3.3	-200
02/04/2004	04:00:00	-200	36	2.8	37
02/04/2004	05:00:00	0.7	51	2.6	56
02/04/2004	06:00:00	1.1	93	4.3	91
02/04/2004	07:00:00	2.6	284	13	159
02/04/2004	08:00:00	3.9	486	20.3	190
02/04/2004	09:00:00	5	798	20.7	249
02/04/2004	10:00:00	3.3	524	15.9	196
02/04/2004	11:00:00	2.9	468	14.2	180
02/04/2004	12:00:00	3.1	454	15.9	167
02/04/2004	13:00:00	3	461	16.1	159
02/04/2004	14:00:00	2.7	391	14.7	152
02/04/2004	15:00:00	2.7	337	14	149
02/04/2004	16:00:00	2.6	297	13.6	143
02/04/2004	17:00:00	3.7	588	19.7	185
02/04/2004	18:00:00	4.5	721	23.3	223
02/04/2004	19:00:00	4.7	710	24.1	215
02/04/2004	20:00:00	5.5	787	25.4	271
02/04/2004	21:00:00	3	415	14.1	180

02/04/2004	22:00:00	1.9	245	10.1	115
02/04/2004	23:00:00	2.3	294	12.1	149
03/04/2004	00:00:00	1.6	139	9	98
03/04/2004	01:00:00	1.3	98	6.3	73
03/04/2004	02:00:00	1.2	88	5.3	69
03/04/2004	03:00:00	0.9	66	3.8	-200
03/04/2004	04:00:00	0.8	57	3	60
03/04/2004	05:00:00	0.8	57	3.3	56
03/04/2004	06:00:00	0.9	60	3.5	73
03/04/2004	07:00:00	2	200	9.5	159
03/04/2004	08:00:00	3	451	15.2	203
03/04/2004	09:00:00	3.1	422	14.3	211
03/04/2004	10:00:00	-200	-200	11.2	-200
03/04/2004	11:00:00	-200	-200	11.4	-200
03/04/2004	12:00:00	-200	-200	13.4	-200
03/04/2004	13:00:00	-200	-200	10.6	-200
03/04/2004	14:00:00	-200	-200	10.8	-200
03/04/2004	15:00:00	-200	-200	11.9	-200
03/04/2004	16:00:00	-200	-200	13.5	-200
03/04/2004	17:00:00	-200	-200	15.2	-200
03/04/2004	18:00:00	-200	-200	17.1	-200
03/04/2004	19:00:00	-200	-200	26.7	-200
03/04/2004	20:00:00	-200	-200	14.2	-200

03/04/2004	21:00:00	-200	-200	8.5	-200
03/04/2004	22:00:00	-200	-200	14.3	-200
03/04/2004	23:00:00	-200	-200	9.3	-200
04/04/2004	00:00:00	-200	-200	7.8	-200
04/04/2004	01:00:00	-200	-200	6.7	-200
04/04/2004	02:00:00	-200	-200	5.4	-200
04/04/2004	03:00:00	-200	-200	5.6	-200
04/04/2004	04:00:00	-200	-200	3.5	-200
04/04/2004	05:00:00	-200	-200	3	-200
04/04/2004	06:00:00	-200	-200	4.4	-200
04/04/2004	07:00:00	-200	-200	3.5	-200
04/04/2004	08:00:00	-200	-200	4.2	-200
04/04/2004	09:00:00	-200	-200	8	-200
04/04/2004	10:00:00	-200	-200	9	-200
04/04/2004	11:00:00	-200	-200	8.1	-200
04/04/2004	12:00:00	-200	-200	8.6	-200
04/04/2004	13:00:00	-200	-200	7.1	-200
04/04/2004	14:00:00	-200	-200	5.6	-200
04/04/2004	15:00:00	-200	-200	10	-200
04/04/2004	16:00:00	-200	-200	10.5	-200
04/04/2004	17:00:00	-200	-200	10.1	-200
04/04/2004	18:00:00	-200	-200	10	-200
04/04/2004	19:00:00	-200	-200	13	-200

04/04/2004	20:00:00	-200	-200	10.5	-200
04/04/2004	21:00:00	-200	-200	5.2	-200
04/04/2004	22:00:00	-200	-200	6.9	-200
04/04/2004	23:00:00	-200	-200	6.4	-200
05/04/2004	00:00:00	-200	-200	4.7	-200
05/04/2004	01:00:00	-200	-200	2.4	-200
05/04/2004	02:00:00	-200	-200	2.2	-200
05/04/2004	03:00:00	-200	-200	1.1	-200
05/04/2004	04:00:00	-200	-200	0.9	-200
05/04/2004	05:00:00	-200	-200	3.1	-200
05/04/2004	06:00:00	-200	-200	6.2	-200
07/04/2004	00:00:00	0.9	93	4	56
07/04/2004	01:00:00	0.7	49	2.3	35
07/04/2004	02:00:00	0.4	30	1.4	21
07/04/2004	03:00:00	0.3	30	0.7	-200
07/04/2004	04:00:00	0.3	30	0.7	12
07/04/2004	05:00:00	0.3	32	0.8	22
07/04/2004	06:00:00	0.8	59	3.6	55
07/04/2004	07:00:00	2.8	277	12.9	152
07/04/2004	08:00:00	3.1	454	15.7	153
07/04/2004	09:00:00	2.3	211	8.6	152
07/04/2004	10:00:00	1.6	196	7.4	112
07/04/2004	11:00:00	1.5	110	6.6	119

07/04/2004	12:00:00	1.6	164	7.8	122
07/04/2004	13:00:00	1.8	151	9.1	115
07/04/2004	14:00:00	1.4	112	6.1	84
07/04/2004	15:00:00	1.9	117	7.5	146
07/04/2004	16:00:00	1.2	120	5.9	65
07/04/2004	17:00:00	2.3	251	12.7	120
07/04/2004	18:00:00	3.3	435	13.5	185
07/04/2004	19:00:00	3.1	345	14.5	157
07/04/2004	20:00:00	3.3	343	11	174
07/04/2004	21:00:00	1.2	62	3.2	62
07/04/2004	22:00:00	1.3	70	4.3	81
07/04/2004	23:00:00	1.9	114	7.1	118
08/04/2004	00:00:00	1.4	88	4.8	95
08/04/2004	01:00:00	1.5	95	5.7	84
08/04/2004	02:00:00	1.1	83	4.1	56
08/04/2004	03:00:00	0.8	63	3.3	-200
08/04/2004	04:00:00	-200	38	1.8	43
08/04/2004	05:00:00	0.8	68	2.8	69
08/04/2004	06:00:00	1.1	59	4.8	74
08/04/2004	07:00:00	2.6	226	13.8	161
08/04/2004	08:00:00	5.1	802	27.1	308
08/04/2004	09:00:00	4.2	585	19.1	300
08/04/2004	10:00:00	2.6	324	11.7	189

08/04/2004	11:00:00	2.4	308	12.7	164
08/04/2004	12:00:00	2.1	238	11.3	122
08/04/2004	13:00:00	2.6	301	14.2	166
08/04/2004	14:00:00	2.8	294	13.9	181
08/04/2004	15:00:00	2.5	353	13.2	127
08/04/2004	16:00:00	1.9	209	10.7	106
08/04/2004	17:00:00	3.6	538	19.7	192
08/04/2004	18:00:00	4.6	808	24	241
08/04/2004	19:00:00	6.3	974	29.1	326
08/04/2004	20:00:00	4.3	544	15.8	232
08/04/2004	21:00:00	1.6	138	7	92
08/04/2004	22:00:00	1.4	92	6.3	95
08/04/2004	23:00:00	2	137	-200	129
09/04/2004	00:00:00	2.4	189	-200	154
09/04/2004	01:00:00	1.8	159	-200	118
09/04/2004	02:00:00	1	80	-200	69
09/04/2004	03:00:00	1	66	-200	-200
09/04/2004	04:00:00	1	87	-200	97
09/04/2004	05:00:00	0.9	79	-200	145
09/04/2004	06:00:00	1.5	150	-200	169
09/04/2004	07:00:00	2.6	196	-200	250
09/04/2004	08:00:00	2.9	299	-200	215
10/04/2004	06:00:00	1.4	134	6	112

10/04/2004	07:00:00	1.7	157	8.2	150
10/04/2004	08:00:00	2	167	9.9	138
10/04/2004	09:00:00	2	157	8.8	124
10/04/2004	10:00:00	1.9	172	8.4	124
10/04/2004	11:00:00	2.1	114	7.5	115
10/04/2004	12:00:00	1.8	166	7.8	101
10/04/2004	13:00:00	1.5	94	6.4	90
10/04/2004	14:00:00	1.3	108	5.1	90
10/04/2004	15:00:00	1.4	98	6.9	89
10/04/2004	16:00:00	1.5	132	6.8	92
10/04/2004	17:00:00	1.7	144	7.4	103
10/04/2004	18:00:00	2	165	8.7	122
10/04/2004	19:00:00	2.5	210	11.2	159
10/04/2004	20:00:00	2.4	277	10.5	172
10/04/2004	21:00:00	2	168	8.2	125
10/04/2004	22:00:00	2.9	248	11.6	187
10/04/2004	23:00:00	2.5	235	9.7	174
11/04/2004	00:00:00	1.4	84	5.1	75
11/04/2004	01:00:00	1.2	75	4	48
11/04/2004	02:00:00	1	62	3.7	61
11/04/2004	03:00:00	1	66	4	-200
11/04/2004	04:00:00	-200	49	3	38
11/04/2004	05:00:00	0.7	38	2.9	38

11/04/2004	06:00:00	0.8	44	2.9	41
11/04/2004	07:00:00	0.9	60	3	51
11/04/2004	08:00:00	1.1	88	3.7	61
11/04/2004	09:00:00	1	70	3.9	47
11/04/2004	10:00:00	1.6	94	6.2	87
11/04/2004	11:00:00	1.8	126	7.3	107
11/04/2004	12:00:00	2.6	181	9.2	161
11/04/2004	13:00:00	1.1	66	4	61
11/04/2004	14:00:00	0.6	54	2.3	26
11/04/2004	15:00:00	0.9	48	3.1	42
11/04/2004	16:00:00	1	66	3.2	55
11/04/2004	17:00:00	1.2	65	3.3	65
11/04/2004	18:00:00	1.4	68	4	75
11/04/2004	19:00:00	2	118	5.7	93
11/04/2004	20:00:00	1.1	65	3.4	63
11/04/2004	21:00:00	1.1	63	3.5	60
11/04/2004	22:00:00	1.1	61	2.9	63
11/04/2004	23:00:00	1.1	63	3.1	66
12/04/2004	00:00:00	0.7	31	1.6	41
12/04/2004	01:00:00	0.6	29	1.1	42
12/04/2004	02:00:00	0.7	34	2	36
12/04/2004	03:00:00	0.6	34	1.7	-200
12/04/2004	04:00:00	0.3	9	0.7	16

12/04/2004	05:00:00	0.3	14	0.5	16
12/04/2004	06:00:00	0.4	23	0.7	33
12/04/2004	07:00:00	0.5	36	0.9	39
12/04/2004	08:00:00	0.6	39	1.5	45
12/04/2004	09:00:00	0.8	38	2.2	63
12/04/2004	10:00:00	1	55	2.9	71
12/04/2004	11:00:00	1.3	75	3.9	104
12/04/2004	12:00:00	1.1	64	2.8	85
12/04/2004	13:00:00	0.6	39	1.4	53
12/04/2004	14:00:00	0.7	42	1.7	58
12/04/2004	15:00:00	1.2	78	3.6	89
12/04/2004	16:00:00	1.4	69	3.1	96
12/04/2004	17:00:00	1.2	67	3.5	88
12/04/2004	18:00:00	1.3	83	3.9	100
12/04/2004	19:00:00	1.8	79	5.2	127
12/04/2004	20:00:00	1.2	68	3.7	88
12/04/2004	21:00:00	0.8	56	2.7	66
12/04/2004	22:00:00	0.8	49	2.6	61
12/04/2004	23:00:00	0.8	56	2.4	54
13/04/2004	00:00:00	0.7	48	2	46
13/04/2004	01:00:00	0.5	27	1.3	25
13/04/2004	02:00:00	0.3	29	1.2	18
13/04/2004	03:00:00	0.4	32	1.3	-200

13/04/2004	04:00:00	0.3	47	0.9	21
13/04/2004	05:00:00	0.7	66	3.1	76
13/04/2004	06:00:00	1.6	163	6.9	149
13/04/2004	07:00:00	3.9	524	19.1	328
13/04/2004	08:00:00	4.5	657	22.1	282
13/04/2004	09:00:00	2.7	324	11.8	206
13/04/2004	10:00:00	1.6	144	7.3	133
13/04/2004	11:00:00	1.6	135	8	137
13/04/2004	12:00:00	1.6	140	7.2	116
13/04/2004	13:00:00	1.5	141	7.7	112
13/04/2004	14:00:00	1.8	181	9.7	128
13/04/2004	15:00:00	2.1	227	9.8	145
13/04/2004	16:00:00	1.7	149	7.7	110
13/04/2004	17:00:00	3	425	15.3	184
13/04/2004	18:00:00	4.6	669	21.5	243
13/04/2004	19:00:00	5	680	21.7	259
13/04/2004	20:00:00	3.5	446	14.6	205
13/04/2004	21:00:00	2.1	205	9.3	144
13/04/2004	22:00:00	2.1	194	9.4	152
13/04/2004	23:00:00	1.5	115	6.8	108
14/04/2004	00:00:00	1.1	74	4.9	64
14/04/2004	01:00:00	0.9	85	3.9	73
14/04/2004	02:00:00	0.7	66	2.7	47

14/04/2004	03:00:00	0.4	60	1.9	-200
14/04/2004	04:00:00	-200	40	1.7	45
14/04/2004	05:00:00	0.9	47	3.3	79
14/04/2004	06:00:00	1.6	116	7.4	113
14/04/2004	07:00:00	3.9	478	18.2	263
14/04/2004	08:00:00	5	836	27.7	275
14/04/2004	09:00:00	4.3	655	18.3	263
14/04/2004	10:00:00	2.7	312	12.3	183
14/04/2004	11:00:00	2.1	195	8.9	147
14/04/2004	12:00:00	2.1	238	10.4	132
14/04/2004	13:00:00	-200	-200	9.7	-200
14/04/2004	14:00:00	-200	-200	12.8	-200
14/04/2004	15:00:00	-200	-200	12.9	-200
14/04/2004	16:00:00	-200	-200	8.9	-200
14/04/2004	17:00:00	-200	-200	11	-200
14/04/2004	18:00:00	-200	-200	12.6	-200
14/04/2004	19:00:00	-200	-200	13.3	-200
14/04/2004	20:00:00	-200	-200	18.1	-200
14/04/2004	21:00:00	-200	-200	13	-200
14/04/2004	22:00:00	-200	-200	7.8	-200
14/04/2004	23:00:00	-200	-200	5.8	-200
15/04/2004	00:00:00	-200	-200	4.7	-200
15/04/2004	01:00:00	-200	-200	3.1	-200

15/04/2004	02:00:00	-200	-200	2.1	-200
15/04/2004	03:00:00	-200	-200	1.7	-200
15/04/2004	04:00:00	-200	-200	1.3	-200
15/04/2004	05:00:00	-200	-200	2.6	-200
15/04/2004	06:00:00	-200	-200	7.9	-200
15/04/2004	07:00:00	-200	-200	21.4	-200
15/04/2004	08:00:00	-200	-200	22.1	-200
15/04/2004	09:00:00	3.9	536	19.1	309
15/04/2004	10:00:00	3.8	481	17.3	327

Table C.1: Experimetal dataset

Table C.2: Analysis of *SISDaV* with Single Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Reasoning Time (sec)				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0.0027	0.0025	0.0019	0.0016	0.0015
Turtle	0.0023	0.0020	0.0016	0.0013	0.0012
NTriple	0.0032	0.0030	0.0023	0.0019	0.0012
N3	0.0022	0.0020	0.0016	0.0013	0.0012

Table C.3: Analysis of *SISDaV* with Ten (10) Inconsistent Data Point per Streaming Window

Format	Cumulative Mean Average of Reasoning Time (sec)				
	400	**800**	**1200**	**1600**	**2000**
RDF/XML	0.0015	0.0012	0.0010	0.0010	0.0009
Turtle	0.0012	0.0010	0.0008	0.0008	0.0007
NTriple	0.0016	0.0013	0.0012	0.0010	0.0010
N3	0.0012	0.0010	0.0009	0.0008	0.0007

Appendix D

Rule Listings for Validation of Indoor Temperature

D.1 Sensor Interference from Heating system

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#


[heatRegulator:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,22)
lessThan(?tempValue,31)
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'With Heater Activated')

]
```

D.2 Sensor Interference from Cooling system

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#


[coolantRegulator:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,11)
lessThan(?tempValue,29)
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'With Coolant Activated')

]
```

D.3 Sensor Interference from Heating and Cooling system

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#


[heatAndCoolantRegulator:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,11)
lessThan(?tempValue,31)
```

```
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'Heater and Coolant Activ

]
```

D.4 Sensor Interference from Outdoor Temp. in Autum

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#

[seasonAutum:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,-15.6)
lessThan(?tempValue,35.7)
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'Season Autum')
]
```

D.5 Sensor Interference from Outdoor Temp. in Spring

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#


[seasonSpring:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
```

```
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,-21.2)
lessThan(?tempValue,32.9)
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'Season Spring')

]
```

D.6 Sensor Interference from Outdoor Temp. in Summer

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix smartSpace: http://localhost:8080/smartSpace#


[seasonSummer:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,-5.7)
lessThan(?tempValue,38.6)
equal(?humidityTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'Season Summer')

]
```

D.7 Sensor Interference from Outdoor Temp. in Winter

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix owl: http://www.w3.org/2002/07/owl#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
```

```
@prefix smartSpace: http://localhost:8080/smartSpace#


[seasonWinter:
(?humidityReadings smartSpace:hasHumidityReading ?humidityValue)
(?humidityReadings smartSpace:humidityHasTimestamp ?humidityTime)
greaterThan(?humidityValue,29)
lessThan(?humidityValue,61)
(?tempReadings smartSpace:tempHasTimestamp ?tempTime)
(?tempReadings smartSpace:hasValue ?tempValue)
greaterThan(?tempValue,-25.3)
lessThan(?tempValue,19.8)
(?pressureReadings smartSpace:hasPressureReading ?pressureValue)
(?pressureReadings smartSpace:PressureHasTimestamp ?pressureTime)
greaterThan(?pressureValue,750.1)
lessThan(?humidityValue,761.0)
equal(?humidityTime,?tempTime)
        equal(?pressureTime,?tempTime)
->
(?tempReadings smartSpace:isValidForPlausibilityCheck 'Season Winter')

]
```