FACULTY OF ENGINEERING AND ARCHITECTURE

Cross Application Interoperability by Management of Smart Home Sensor Data in Solid

Wout Slabbinck
Student number: 01405025

Supervisors: Prof. dr. ir. Jeroen Hoebeke, Prof. dr. ir. Ruben Verborgh
Counsellors: Bart Moons, Dr. Pieter Colpaert

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2020-2021

GHENT UNIVERSITY

Acknowledgment

This year, I learned a lot due to this thesis. Apart from a deep dive into the Internet of Things and the technologies which make it possible to surf the Web, I learned a lot about working regularly to complete a big project. That is why I would like to thank several people.

First, I would like thank my supervisors Prof. dr. ir. Jeroen Hoebeke and Prof. dr. ir. Ruben Verborgh for the opportunity to work on and experimenting with Solid, combined with my passion for Internet of Things. And for the many meetings, where they posed numeral questions which allowed me to design a better solution than I had initially designed.

Furthermore, I would like to thank Bart Moons and Dr. Pieter Colpaert for everything they have done during the year counselling me. For every question I had, they tried to form an answer or lead me to the right direction to find the answer myself. I am especially thankful that I could contact them whenever I had a problem.

I am also grateful for allowing me to continue working on this topic as a starting doctorate student.

I would also like to thank my parents as they were always open to listen to the problems I faced during the dissertation process. They also gave useful feedback as outsiders to the topic, which allowed me to zoom out and see the bigger picture from time to time. Finally, I wish to thank my girlfriend, Anaïs, who kept me motivated during the difficult times during my thesis.

Cross Application Interoperability by Management of Smart Home Sensor Data in Solid

Wout Slabbinck
Supervisors: Prof. dr. ir. Jeroen Hoebeke, Prof. dr. ir. Ruben Verborgh
Counsellors: Bart Moons, Dr. Pieter Colpaert

***Abstract*—Smart homes consist of a collection of Internet of Things (IoT) devices, often from different domains. Due to those heterogeneous domains, it is hard to integrate all the smart devices into one solution. Furthermore, it is hard to share data of an IoT solution due to its monolithic nature. In this paper, a solution is proposed to make the underlying data model interoperable by defining a shape. Only data conforming to that shape can be stored on a Linked Data Platform supporting the novel Shape Trees specification. Proof-of-concept results show that the data validation process has a negative impact on the time it takes to store data to the platform. The delay is proportional to the volume of the stored data. Therefore, a fragmentation service and retention service were implemented, which counteracted the validation delay. Future work will try to find a dynamic optimal fragmentation strategy.**

***Index Terms*—Internet of Things (IoT), Linked Data Platform (LDP) , Solid, TREE, Shape Trees**

I. Introduction

The number of smart homes grows rapidly. This implies that there are a lot of smart devices integrated in homes, which are connected to the internet. This connectivity enables services to interact with those smart devices and the interconnection of all those devices forms the Internet of things (IoT). In the IoT there are multiple application domains, each with their own solution. For example, the energy domain has a solution to control your lights via the internet or the security domain has an application to remotely monitor the alarm status.

However, controlling each solution is done with a vendor-specific controller [1] and often there is no or limited interoperability across systems. This problem is known as vertical silos and is common in the IoT sector [2].

Furthermore, sharing a subset of sensor data is not straightforward, as an IoT solution is generally a monolithic system, meaning that either everything is shared or nothing is shared. In such solution, shared access to all devices is possible either through multiple accounts, or by the same credentials to be shared by multiple users. This is considered bad practice as it forms a major security risk [3]. Yet, sharing a subset of sensor data can be highly desired. For instance, for public services end-users might want to share the alarm status of their security system with the police or the temperature sensor data from the rooms in their house with the fire department.

In order to build cross-domain services and platforms that have restricted access without having an impact on the user experience and the location of data storage, several requirements should be fulfilled. First, the data must be defined with an interoperable data model. It must also be possible to share data and to define access control policies.

Therefore, this paper introduces an architecture to meet these requirements. A prototype has been built and the architectural design was validated.

II. Background

This section briefly describes the technologies used for building the prototype.

Linked Data (LD) [4] is used to accomplish interoperability. It is a graph-based way to structure data and is used to give meaning to data. To represent Linked Data, the Resource Description Framework (RDF) [5] is used. In Linked Data, multiple vocabularies or Ontologies exist which define relationships and concepts to describe a certain topic.

The *Semantic Sensor Network* (SSN) Ontology [6] was created to describe sensors and its metadata. The Sensor, Observation, Sample, and Actuator (SOSA) ontology, the core of SSN, is used to model the sensor data.

Linked Data Platform (LDP) [7] is a W3C standard that defines rules to exchange Linked Data on the web using HyperText Transfer Protocol (HTTP) operations.

The *Shape Expressions* (ShEx) language [8] is used to describe and validate RDF graphs in a machine-readable way.

The *Shape Trees* [9] specification defines the resources that an LDP can contain. Resources with an RDF format are validated by a shape file. Enforcing valid resources facilitates interoperability.

The *Social Linked Data* (Solid) Protocol [10, 11] is a collection of specifications that define a platform for decentralized data storage. The data is stored to the personal online datastore (pod) of the end user. In Solid, LDP is used as a platform, for authentication the Web Identity and Discovery (WebID) [12] is used in combination with Solid-OpenID Connect (OIDC) [13] and authorization is defined by the Web Access Control (WAC) specification [14].

The *TREE Hypermedia* specification [15] is used to define a fragmentation strategy to avoid that all the sensor data on the LDP needs to be stored in one resource.

The *Linked Data Event Stream* (LDES) specification [16] is an extension of TREE. It is used to define the retention policy, which is defined to control the storage volume.

The *RDF Mapping Language* (RML) [17] defines mapping rules to convert JavaScript Object Notation (JSON) files to RDF resources.

Long Range (LoRa) *Wide Area Network* (WAN) [18] is a long range low-power communication technology standard, used to transmit the sensor data to a broker.

The *Message Queuing Telemetry Transport* (MQTT) protocol [19] is a lightweight publish-subscribe broker. Data generating machines publish data to the broker and services subscribe to the broker to get notified by an event to receive data.

III. Challenges

This section presents the two major challenges that had to be overcome to create the prototype. Also, several solutions to solve these challenges are given.

A. RDF Validation

Defensive programming is generally used when consuming data received from another service to make sure a procedure will work as intended. A downside of defensive programming is that it is time consuming and exhaustively defining all cases is quite complex. In order to guarantee that a certain graph in an RDF resource will follow a specific model, the W3C created Shapes for RDF. This has the advantage that no custom SPARQL queries should be written anymore to validate the expectations that certain elements were indeed in the RDF resource [20].

However, the task to manually validate every graph that is used in a program, even with shapes, is quite labour intensive. For this reason, the Shape Tree specification [9] was created to validate graphs for LDPs.

No research has yet been done to verify if validation has an impact on the performance of an LDP.

B. HTTP PATCH requests

An implementation for adding sensor data to a Solid Pod was already presented in [21]. However, the author noticed that as files grew in data volume, the delay of PATCH requests increased linearly with growth in data volume.

To overcome the above issue, two strategies are proposed.

1) Fragmenting: A first solution fragments the file into multiple smaller files. The idea is that the time required to both validate and add new measurements should be significantly lower when using smaller files.

2) Retention policy: The second solution defines a retention policy. The impact of such a policy is that the total volume of the stored file remains limited, which should positively affect the validation and patch times.

IV. Architecture

A prototype was built that meets the requirements defined in the introduction. The architecture of this prototype is shown in Figure 1.

It consists of several LoRa sensors that generate data. This data is transferred via several gateways to a service (RML mapper) which transforms the data to Linked Data. This service also stores the data to the LDP which validates the data. Finally, valid data is stored on a Solid Pod.

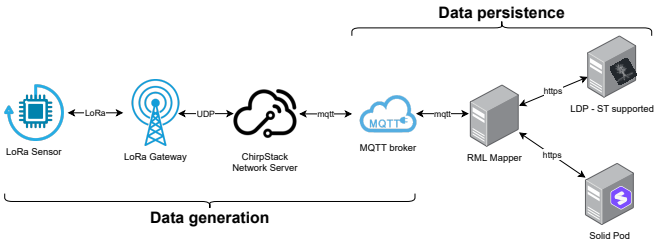

Fig. 1. Architecture

A. Data generation

The sensor devices measure different properties from their environment such as temperature, humidity, pressure... Every ten minutes, a packet containing these measurements is transmitted to a LoRa Gateway. The LoRa Gateway is configured to send the data from the registered sensors to the ChirpStack Gateway Bridge, a component of the ChirpStack project [22]. The ChirpStack project is an open-source solution to manage LoRaWAN networks. The ChirpStack Gateway Bridge publishes the sensor data to the MQTT broker to which the ChirpStack Network Server (NS) is subscribed. The NS deduplicates these messages and publishes them to the MQTT broker, to which the RML Mapper is subscribed.

B. Data persistence

When an event is received from the MQTT broker, the RML Mapper decodes the sensor data to an intermediary representation. The service then transforms the sensor data to Linked Data and transmits it over HTTPS to the LDP server, which has Shape Trees support. After validating the data and storing it in the LDP, the RML Mapper finally puts the Linked Data in a Solid Pod. Data is not directly stored on the Solid Pod, as Solid does not support Shape Trees. Finally, the storage size of the Solid Pod is controlled by a retention policy to ensure scalability using LDES. The fact that LDES is an extension of TREE allows defining a fragmentation strategy by introducing a collection of items that can be fragmented, where the fragments are interlinked.

V. Evaluation

The proposed solution is evaluated in terms of scalability. The evaluation verifies if fragmentation indeed reduces the time to add new observations to the LDP. Furthermore, the impact of preserving validated files on the LDP that keeps growing in volume, is quantified.

The test setup consists of 2 servers. One server hosts the LDP with Shape Tree support, the other hosts the RML Mapper.

The performance of the solution is tested by comparing three configurations. The first configuration stores the sensor data using the SOSA model to a single file on the LDP. The second configuration also stores data to a single file, however, now represented by a TREE collection using the SOSA model. The final configuration uses the same strategy as the second one utilizing a fragmentation strategy at a given interval.

A. Performance of fragmentation strategy on adding data

In order to compare the three configurations, the RML Mapper must handle the same amount of messages from the broker.

A test was executed by publishing 50 measurements to the broker with a delay of 10 seconds between consecutive measurements. Figure 2 shows how a growing file size impacts the delay of a PATCH request. When the stored size surpasses 400 kB, adding new measurements to the plain SOSA model takes more than 9 seconds. On the contrary, as shown in Figure 3, a PATCH request to a TREE collection takes only 1 second.

The difference in performance between adding measurements to plain SOSA or to a TREE collection is mainly caused by a more complex validation strategy for the single SOSA model. Though, the number of measurements continues to grow over time. As the file size increases, adding new measurements will take more time as well. Hence, appending measurements to a single file without a further optimization is not feasible.

Therefore, fragmentation strategy was implemented that executes every 10 seconds. Figure 4 shows that such a fragmentation strategy can drastically decrease the delay of a PATCH request. As a result, the stored file size is now limited to 120 kB and a single PATCH request only takes 0.35 seconds.

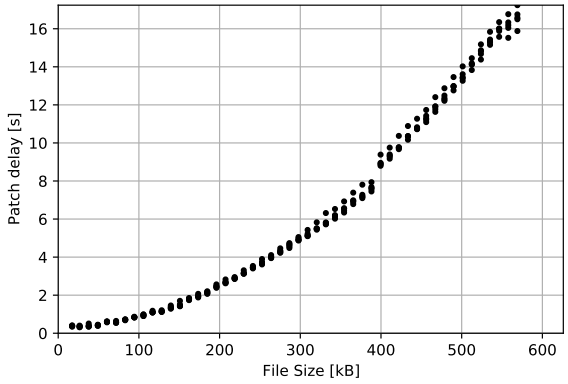

Fig. 2. SOSA model: stored file size versus delay

B. Analysis of validation on storing data

During performance tests, a notable difference was observed when adding data to the LDP with the SOSA model versus the TREE collection using SOSA.

In order to test whether validation has an impact when adding new measurements, a burst mode test was executed by publishing 50 measurements to the broker with a delay of 1 second between consecutive measurements.

The results showed that when no validation is executed by the LDP, it takes the same amount of time to PATCH data for either configuration. Both were executed within 0.7 seconds, proving that validation has a big impact on storing to one file.

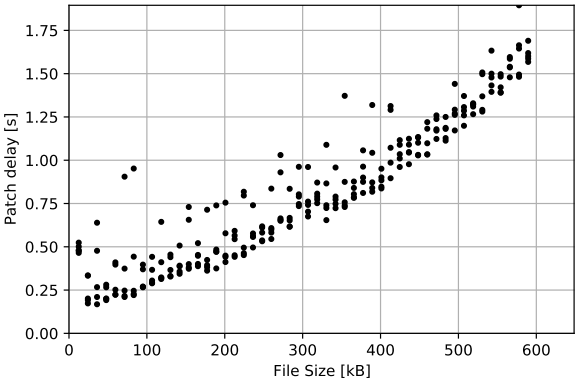

Fig. 3. TREE collection with SOSA model: stored file size versus delay

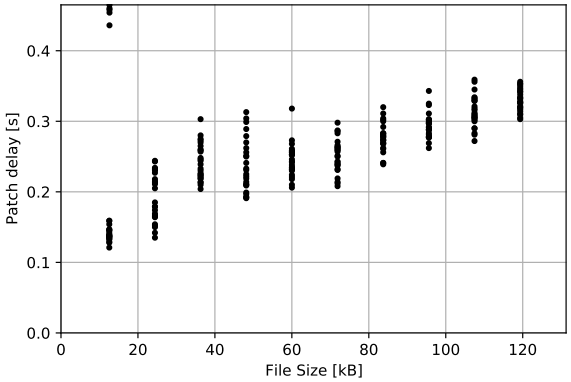

Fig. 4. TREE collection with SOSA model: stored file size versus delay, fragmenting every 10 measurements

VI. Conclusion

In this paper, a solution was designed for storing sensor data in an interoperable way to a Linked Data Platform. Storing the data to Solid, a novel type of LDP, allows the end user to have full control over that data. Furthermore, the solution was further extended to allow the volume of sensor data to scale in size. An architecture based on open standards has been designed and implemented. This involved processing steps from data generation to storage, together with two services. The first service, the fragmentation service, improves the speed of storing new measurements to the dataset. The retention service manages the total volume of the dataset by executing user defined policies.

The fragmentation service was evaluated and was shown to be mandatory for handling and storing a large amount of measurements.

The need for retention policies was clarified. Evaluation of the retention policy showed that when the maximum amount of observations was reached, the data volume would not grow above a certain size.

VII. Future work

Future work might include the following:

- adapt the current implementation to store data directly to a Pod.
- add support for handling *burst data*. A suggestion is to aggregate data over a time window of 5 minutes and then update the LDP with this aggregated data.
- improve the retention service by storing the total amount of members per fragment in the root node.
- enforce members in an even stream to be immutable, to conform to the LDES specification
- find a general and dynamic formula to define the fragmentation size.

References

[1] J.-L. Gassée, "Internet of Things: The "Basket of Remotes" Problem," Mar. 2016. [Online]. Available: https://mondaynote.com/internet-of-things-the-basket-of-remotes-problem-f80922a91a0f

[2] F. Cirillo, F.-J. Wu, G. Solmaz, and E. Kovacs, "Embracing the Future Internet of Things," *Sensors*, vol. 19, no. 2, p. 351, Jan. 2019. [Online]. Available: http://www.mdpi.com/1424-8220/19/2/351

[3] Z. Aufort, "Sharing Credentials is a Really Bad Idea. Seriously, Don't Do It." Oct. 2020. [Online]. Available: https://puget.tech/sharing-credentials-bad-idea/

[4] T. Berners-Lee, "Linked Data - Design Issues," Jul. 2006. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[5] O. Lassila and R. R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification." [Online]. Available: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[6] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic Sensor Network Ontology," Oct. 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/

[7] S. Speicher, J. Arwe, and A. Malhotra, "Linked Data Platform 1.0," Feb. 2015. [Online]. Available: https://www.w3.org/TR/ldp/

[8] E. Prud'hommeaux, I. Boneva, J. E. L. Gayo, and G. Kellog, "Shape Expressions Language 2.1." [Online]. Available: http://shex.io/shex-semantics/

[9] E. Prud'hommeaux and J. Bingham, "Shape Trees Specification." [Online]. Available: https://shapetrees.org/TR/specification/

[10] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, J. Bingham, and D. Zagidulin, "The Solid Protocol." [Online]. Available: https://solidproject.org/TR/protocol#authentication

[11] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: A Platform for Decentralized Social Applications Based on Linked Data," p. 16, 2016.

[12] A. Sambra, H. Story, and T. Berners-Lee, "WebID 1.0," Mar. 2014. [Online]. Available: https://www.w3.org/2005/Incubator/webid/spec/identity/

[13] A. Coburn, E. Pavlik, and D. Zagidulin, "SOLID-OIDC." [Online]. Available: https://solid.github.io/authentication-panel/solid-oidc/

[14] "Web Access Control (WAC)." [Online]. Available: http://solid.github.io/web-access-control-spec/

[15] P. Colpaert, "The TREE hypermedia specification," Feb. 2021. [Online]. Available: https://treecg.github.io/specification/

[16] ——, "Linked Data Event Streams spec," Feb. 2021. [Online]. Available: https://semiceu.github.io/LinkedDataEventStreams/eventstreams.html

[17] A. Dimou, M. Vander Sande, B. De Meester, P. Heyvaert, and T. Delva, "RDF Mapping Language (RML)." [Online]. Available: https://rml.io/specs/rml/

[18] N. Sornin, M. Luis, T. Eirich, and T. Kramp, "LoRaWAN® Specification v1.1." [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/

[19] "MQTT - The Standard for IoT Messaging." [Online]. Available: https://mqtt.org/

[20] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 7, no. 1, pp. 1–328, Sep. 2017. [Online]. Available: http://www.morganclaypool.com/doi/10.2200/S00786ED1V01Y201707WBE016

[21] F. Sanders, "Solid Pods for IoT," 2021.

[22] "ChirpStack open-source LoRaWAN® Network Server." [Online]. Available: https://www.chirpstack.io/

Contents

ListofFigures **13**

1 Introduction **19**

2 Related work **21**

2.1 Semantics 21

2.1.1 Linked Data 22

2.1.2 Resource Description Framework 23

2.1.3 Ontologies 24

2.1.4 Querying 27

2.2 Linked Data Platform 29

2.3 Shapes 32

2.3.1 Shape Expressions 32

2.3.2 Shapes Constraint Language 32

2.3.3 ShEx Example 33

2.4 Shape Trees 34

2.5 Social Linked Data 37

2.6 Fragmentation strategy 38

2.6.1 TREE hypermedia 38

2.6.2 Linked Data Event Streams . . . 39
2.7 RDF Mapping Language . . . 41
2.8 Sensor network and communication . . . 41
2.9 Broker technology . . . 42
2.10 Challenges . . . 42
2.10.1 RDF validation . . . 42
2.10.2 PATCH requests . . . 43

3 Architecture **44**
3.1 Data generation . . . 45
3.2 Data persistence . . . 45
3.3 Fragmentation and retention service . . . 46

4 Implementation **49**
4.1 Converting sensor data to Linked Data . . . 49
4.2 Adding new data to the LDP . . . 50
4.2.1 Adding SOSA sensor data to an LDP with Shape Tree Support . . . 51
4.2.2 Adding SOSA sensor data as part of a TREE collection to an LDP with Shape Tree support . . . 52
4.3 Fragmentation . . . 52
4.4 Retention policy . . . 53
4.5 Copying to a Solid Pod . . . 56
4.6 Editing the access control list for a collection . . . 56

5 Evaluation **62**
5.1 Setup . . . 62

5.2 Performance of fragmentation strategy on adding data 63

5.2.1 Analysis for the optimal fragmentation strategy 64

5.3 Analysis of validation on storing data 66

5.4 Impact of the retention policy 67

6 Conclusion **70**

7 Future work **72**

7.1 Shape Tree supported Solid Pod 72

7.2 Burst data 72

7.3 Improve the retention service 73

7.4 Enforce members in an event stream to be immutable 73

Bibliography **74**

Appendices **80**

ListofFigures

2.1 Triple 23

2.2 SOSA and SSN ontologies and their extensions, adopted from [1] 25

2.3 Samples of different types of LDPRs, adopted from [2] 29

2.4 Hierarchy of the LDPRs, adopted from [2] 30

2.5 LoRaWAN architecture: star of stars topology, adopted form [3] 42

3.1 Architecture 44

3.2 Sequence diagram of the flow from sensor to the MQTT broker 45

3.3 Sequence diagram of storing the sensor data to the Solid Pod 46

3.4 Storage structure on the LDP 47

3.5 Fragmentation storage structure on the LDP 48

3.6 Sequence diagram of the fragmentation service and the retention policy execution 48

5.1 SOSA model in burst mode 63

5.2 TREE collection with SOSA model in burst mode 63

5.3 TREE collection with SOSA model in burst mode, fragmenting every 10 measurements 63

5.4 SOSA model: stored file size versus delay 65

5.5 TREE collection with SOSA model: stored file size versus delay 65

5.6 TREE collection with SOSA model: stored file size versus delay, fragmenting every 10 measurements 65

5.7 Every 10 additions fragmentation strategy: stored file size versus delay 67

5.8 Every 20 additions fragmentation strategy: stored file size versus delay 67

5.9 Every 50 additions fragmentation strategy: stored file size versus delay 67

5.10 Once per day fragmentation strategy: stored file size versus delay 67

5.11 Fragmentation times versus total number of measurements in the LDP for the different fragmentation configurations 68

5.12 SOSA model in burst mode without validation 69

5.13 Tree collection with SOSA model in burst mode without validation 69

5.14 Tree collection with SOSA model in burst mode without validation, fragmenting every 10 measurements 69

5.15 File size versus patch delay while running the retention policy every 10 measurements 69

5.16 File size versus patch delay while running the retention policy every 10 measurements 69

List of Listings

2.1 Triple in Turtle format 23
2.2 Triple in turtle format with prefixes and a base 24
2.3 Complete SOSA example 25
2.4 A SPARQL query on SOSA data 27
2.6 LDP Basic container 29
2.5 HTTP GET request to an LDPC 30
2.8 LDP Basic container, after adding example1.ttl 30
2.7 HTTP POST request to an LDPC 31
2.9 Shape Expression for a temperature sensor in shexC 33
2.10 Shape Tree in Turtle format 35
2.11 Shape Trees Plant operation to a local hosted LDP that supports Shape Trees . 35
2.12 Shape Trees Create Data Instance operation 36
2.13 A collection of observations with a root node of a TREE fragment 38
2.14 A collection of observations of a TREE fragment 39
2.15 A LDES collection of observations with a retention policy 40
4.1 HTTP PATCH request to the LDP 51
4.2 An example of a time-based retention policy 54
4.3 An example of the of a version-based retention policy 54
4.4 Read acces to a resource 56
5.1 Version based policy that allows for 500 measurements to be stored 67

Acronyms

ACL Access Control List. 37, 56, 57, 73

CRUD Create, Read, Update, and Delete. 29, 31

CSV Comma-Separated Values. 41

DNS Domain Name System. 81

FTP File Transfer Protocol. 22

HTML HyperText Markup Language. 22

HTTP HyperText Transfer Protocol. 22, 23, 29–31, 35, 36, 50, 51, 81

HTTPS HyperText Transfer Protocol Secure. 45, 81

IoT Internet of Things. 19

IP Internet Protocol. 81

JSON JavaScript Object Notation. 41, 49, 82, 84

JSON − LD JavaScript Object Notation for Linked Data. 23

LDES Linked Data Even Streams. 20, 40, 47

LDF Linked Data Fragments. 28

LDP Linked Data Platform. 11, 20, 29, 34–37, 43–47, 50–52, 56, 62–66, 68, 70, 72, 73

LDP − BC LDP Basic Container. 29, 30

LDP − NRS LDP Non-RDF Sources. 29

LDP − RS LDP RDF Source. 29

LDPC LDP Container. 29

LDPR LDP Resource. 29, 31, 35

LoRa Long Range. 41, 45

LoRaWAN Long Range Wide Area Network. 41

MQTT Message Queuing Telemetry Transport. 42, 45, 49, 82

N3 Notation3. 23

NB − IoT NarrowBand IoT. 41

NS Network Server. 41, 42, 45

NSS Node Solid Server. 45, 56

OGC Open Geospatial Consortium. 25

OIDC OpenID Connect. 37

OM Ontology of Units of Measure. 25

OSLO Open Standards for Linked Organizations. 25

QUDT Quantities, Units, Dimensions and Data Types. 25, 49, 84

R2RML RDB toRDF Mapping Language. 41

RDF Resource Description Framework. 22–24, 28–35, 38, 41, 42, 46, 105

RML RDF Mapping Language. 41, 45–47, 49, 62, 63, 72, 84

SAREF Smart Appliances REFerence. 26

SHACL Shapes Constraint Language. 32, 40

ShEx Shape Expressions. 32–35, 50

Solid Social Linked Data. 12, 20, 34, 37, 43–45, 62, 70, 72, 73

SOSA Sensor, Observation, Sample, and Actuator. 24–27, 32, 34, 41, 49–52, 62–64, 66, 84, 90, 97, 105, 109, 114

SPARQL SPARQL Protocol RDF Query Language. 22, 27, 28, 32, 42

SSH Secure Shell. 22

SSN Semantic Sensor Network. 24–26

TD Thing Description. 26

TNO Netherlands Organisation for Applied Scientific Research. 26

Turtle Terse RDF Triple Language. 23, 24, 31, 49, 90, 97, 105

UML Unified Modeling Language. 45

URI Unique Resource Identifier. 20, 22–24, 26, 29

URL Uniform Resource Locator. 22, 27, 50, 51, 56, 57

URN Uniform Resource Name. 22

UUID Universally Unique Identifier. 84

W3C World Wide Web Consortium. 25–27, 29, 32, 34, 37, 41

WAC Web Access Control. 37, 56, 73

WebID Web Identity and Discovery. 37, 56, 57

WoT Web of Things. 26

WWW World Wide Web. 22

1

Introduction

Today, more homes are starting to integrate smart devices into their environment. Smart devices are able to sense and act in that environment and are connected to the Internet. This connection makes it possible to use services to interact with your smart devices. The interconnection of all those devices forms the Internet of Things (IoT). In the IoT, a lot of different domains are present. In a home environment for example, there are devices, called sensors, which measure temperature and humidity. Other devices, called actuators, are used to control lights and other electrical appliances, in other words the energy domain. IoT solutions also exist for other domains, for instance home security systems.

However, some problems arise when trying to build services which incorporate all of those domains. For example, when trying to combine all the smart devices into one solution, which can be controlled with a single dashboard. First of all, the different domains have their own ecosystem and thus there is no interoperability across them. Moreover, there are many competitors in each domain, each having their own proprietary solution. For similar solutions, there is often no or limited interoperability across systems. This problem is known as vertical silos and is common in the IoT sector [4]. Currently, this results in the need to monitor and control your smart home with the use of the multiple dashboards from all the different vendors [5].

Furthermore, sharing a subset of sensor data is not straightforward, as an IoT solution is generally a monolithic system, meaning that either everything is shared or nothing is shared. In such

solution, shared access to all devices is possible either through multiple accounts, or by the same credentials to be shared by multiple users. This is bad practice, as it forms a major security risk [6]. Yet, sharing a subset of sensor data is important for some public services like sharing the alarm status with the police or the temperature sensor data with the fire department.

In order to build cross-domain services and platforms that have restricted access without having an impact on the user experience and the location of data storage, several requirements should be fulfilled. First, the data must be defined with an interoperable data model. Standard interfaces must be used to store the data. It must also be possible to share data and to define access control policies. To meet these requirements, this master thesis proposes the following solution. First sensor data is transformed to Linked Data, which is a graph-based way to structure data using Unique Resource Identifiers (URIs) to present data and relations between data. Then the data is stored in a Linked Data Platform (LDP), capable of reading and writing Linked Data. The linked sensor data together with the access control information resides in a Social Linked Data (Solid) [7] Pod and can only be accessed via a WebID [8], which is used for authentication. The Solid ecosystem provides fundamental affordances for decentralised Web applications for information exchange in a way that is secure and privacy respecting.

To explore the capabilities of the Solid ecosystem for sensor data, a prototype is developed which only allows valid data to be stored. The Shape Trees [9] specification guarantees that on the platform only valid files are present. This indicates that the other services know what data to expect, improving the interoperability. Unfortunately, validation is an expensive operation in the sense that it would take a significant amount of time to add new data. A fragmentation strategy is build with the TREE Hypermedia specification [10] to mitigate this validation delay. Additionally with Linked Data Even Streams (LDES) [11], a retention policy service is defined to guarantee that the data volume does not grow infinitely.

To evaluate the prototype, first a test should be performed to see how long it takes to validate the data when more sensor data is added over time, without the two proposed services. Then to verify that validation is an expensive operation, the same test has to be completed without validation. Finally, the performance of the solution with two services has to be compared to the solution where they are not activated to verify that they indeed reduce this validation time.

The outline of this dissertation is as follows: Chapter 2 gives an introduction to the concepts and the technologies used. Chapter 3 describes the design of the solution and explains the design choices. Chapter 4 states the challenges faced during the implementation and how they were solved. Chapter 5 presents and analyzes the results of the solution. Chapter 6 formulates the general conclusion. Chapter 7 focuses on further improvements and possible experiments that would develop more insights.

2

Related work

2.1 Semantics

Semantics, coming from the Ancient Greek word *sēmantikós* [12], which translates to "significant", is the study of meaning, reference, or truth. Semantics are important as machines, more specifically the applications or services, need to be able to interpret the data generated by smart devices. In other words, different devices and machines become interoperable when they agree upon semantics [13]. For example if a device, a sensor, measures the temperature on the 16th of april at 15:00 UTC+2[1], it should be stored in such a way that another service can reason over the data source and knows that it was generated at 15:00. On top of that, there are different temperature scales: Celcius, Fahrenheit, and Kelvin. In order to interpret the temperature, the scale used has to be known. This means all the data should be encoded in such a way that its semantics, the metadata, which in itself means data describing data, are stored together with the data.

[1]UTC is the Coordinated Universal Time, a standard for time. The +2 indicates that the time zone where the example was measured, is 2 hours ahead of the UTC, which corresponds with the summer time of Belgium.

2.1.1 Linked Data

Linked Data facilitates giving meaning to the data. It is a concept for structuring data by interlinking machine-readable data on the web.

Tim Berners-Lee, the founding father of the World Wide Web (WWW) , introduced Linked Data where he proposed four rules [14]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. so that they can discover more things.

The first rule means that things should be made identifiable with the use of a Uniform Resource Identifier (URI) [15]. There are two types of URIs: the Uniform Resource Name (URN) [16] and the Uniform Resource Locator (URL) [17]. The first type is a URI that labels a resource with a persistent, location-independent unique identifier. A URN has the following syntax: *"urn:"* *<NID>* *":"* *<NSS>*, where NID is the Namespace Identifier and NSS is the Name Specific String. The latter is a URI that starts with describing the protocol on how to act on or obtain that resource, the scheme, followed by the unique identification and location of the resource, the scheme specific part. So, the syntax of a URL is: *<scheme>:<scheme-specific-part>*. Some examples of valid protocols are: *mailto*, *ftp*, *ssh*, ... [2]

The measurement given as example at the beginning of this section could be identified with a URN as *urn:thesis:woutslabbinck:chapter2:example:1* and with a URL as *https://woutslabbinck.com/thesis/chapter2/example1*.

The second rule states that the protocol used for the URI should be HTTP (HyperText Transfer Protocol) , a request-response protocol to transfer representation of resources from a server to a client. In Appendix A, the whole syntax of an HTTP URL is described.

The next rule states that after a lookup, the URI is now dereferenced. This dereferencing should provide more useful information about the resource using standards as RDF or SPARQL, which will be explained later in section 2.1.2 and 2.1.4. *Useful* means that the information provides context or an explanation for the resource.

[2] mailto is a scheme that allows users to send a mail to a specific address directly by following the link, often used in HTML web pages. FTP (File Transfer Protocol) is a protocol used for transferring files from a server to a client. And the SSH protocol (Secure Shell Protocol) is a protocol for secure remote communication.

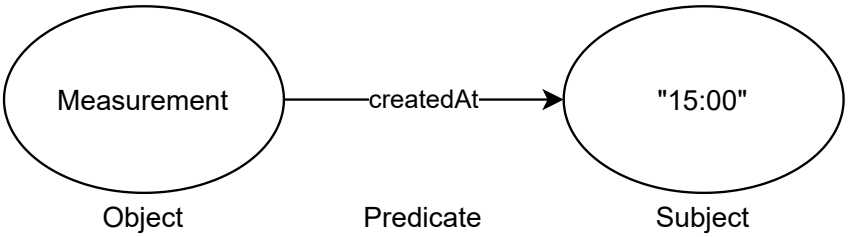

Figure 2.1: Triple

The final rule states that it is possible to link documents together to make a connected knowledge graph of Linked Data. In the example, this rule could be used to give to the value 15:00 the meaning that it stands for time.

Using these aforementioned rules, the relation that a measurement is created at that certain time can be modeled to a triple. A triple consists of three parts: the subject, the predicate and the object. In this example the subject is the measurement, the predicate is the relation that it was created at a certain time and the time is the object. This simple Linked Data triple is visualised in Figure 2.1.

However, this triple is not a true Linked Data triple as the object, predicate and subject are no HTTP URIs, which also means that the predicate and subject do not provide useful information.

When `https://woutslabbinck.com/thesis/chapter2/example1.ttl#measurement1` is used as an object, `https://www.w3.org/ns/sosa/resultTime` as a predicate and ”2021-04-16T15:00:00”^^`http://www.w3.org/2001/XMLSchema#dateTime` as a subject, a correct triple is formed, satisfying the four rules.

2.1.2 Resource Description Framework

In the previous section, the four rules of Linked Data were defined and explained. The third rule stated that using the standards, it is possible to provide useful information when the link is followed. The Resource Description Framework (RDF) [18] is such a framework. RDF is the standard model used for representing the subject-predicate-object triples. There exist multiple serialization formats, which makes it possible to encode them into files, such as Terse RDF Triple Language (Turtle) [19], JavaScript Object Notation for Linked Data (JSON-LD) [20], Notation3 (N3) [21], RDF/XML [22], …. But from now in this thesis, when triples are described, the turtle format is used as it is the most human-friendly format.

The simple example would be represented in Turtle as shown in Listing 2.1.

```
1 <https://woutslabbinck.com/thesis/chapter2/example1.ttl#measurement1> <https://www.w3.org/ns
    /sosa/resultTime> "2021-04-16T15:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

Listing 2.1: Triple in Turtle format

It seems like this is not that readable, but in Turtle it is possible to work with prefixes. Prefixes, often defined at the top of a document, contain a shorthand notation for URIs. It consists of 3 parts: the prefix indicator (@prefix), the prefix label and the URI. The prefix label can then be used by placing it in front of the URI path or fragment. The subject can also be made a relative URI as the location where it is hosted is already known in the document. To indicate the full URI, the @base directive can be used, which is syntactic sugar to resolve the relative URI to the base URI.

An example with prefixes and a base directive clarifies these concepts. Listing 2.2 is a turtle document which is equivalent to Listing 2.1.

```
1 @base <https://woutslabbinck.com/thesis/chapter2/example1.ttl#> .
2 @prefix sosa: <https://www.w3.org/ns/sosa/> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 <measurement1> sosa:resultTime "2021-04-16T15:00:00"^^xsd:dateTime .
```

Listing 2.2: Triple in turtle format with prefixes and a base

2.1.3 Ontologies

With RDF, very detailed, complex and big documents can be created. When a new kind of structure is needed, it does not have to be defined in the document itself. As predicates, the relations, are in itself URIs, they can be defined somewhere else and still be used as intended in the document.

In Listing 2.2, this concept is already used. For the predicate *sosa:resultTime*, a predicate from `https://www.w3.org/ns/sosa/` is used. The structure of this context is thus externally defined. Such a building block, like SOSA, is called an Ontology or vocabulary.

Ontologies are a structural model, often used for one particular context. They are used as building blocks for creating data sets of Linked Data.

Semantic Sensor Network Ontology

To model measurements of sensors using Linked Data, the Semantic Sensor Network (SSN) Ontology [1] is used. This ontology, developed by the Spatial Data on The Web Working Group

Figure 2.2: SOSA and SSN ontologies and their extensions, adopted from [1]

[23], which is a joint Working Group between World Wide Web Consortium (W3C) [24] and Open Geospatial Consortium (OGC) [25], was created to describe sensors, their observations and the interfaces on how to interact with them. The core of SSN is the Sensor, Observation, Sample, and Actuator (SOSA) ontology. SOSA is the main building block of SSN and is intended for light-weight and standalone use[26]. In Figure 2.2, the relation between SOSA, SSN and its extensions can be seen.

Listing 2.3 provides an example of a temperature sensor that made a measurement. This is clarified by the predicate *sosa:madeObservation* from *temperatureSensor observation1* and in the opposite direction by *sosa:madeBySensor*. An observation has a result, which is modeled by external vocabularies which are designed to model quantifiable values. Two such candidates that are often used in SOSA are the Ontology of Units of Measure (OM) [27] and the Quantities, Units, Dimensions and Data Types (QUDT) Ontologies [28]. QUDT is used because these ontologies are the prefered units of measurement in OSLO [29, 30]. The result of the example now has a numeric value as well as a scale indicating the actual temperature measured by the sensor.

```
1 @base <https://woutslabbinck.com/thesis/chapter2/example1.ttl#> .
2 @prefix sosa: <http://www.w3.org/ns/sosa/>.
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
4 @prefix qudt: <http://qudt.org/schema/qudt/>.
5 @prefix unit: <http://qudt.org/vocab/unit/>.
6
7 <temperatureSensor> a sosa:Sensor;
8     sosa:madeObservation <observation1>.
```

```
9
10 <observation1> a sosa:Observation ;
11   sosa:hasResult <result1>;
12   sosa:madeBySensor <temperatureSensor>;
13   sosa:resultTime "2021-04-16T15:00:00"^^xsd:dateTime.
14
15 <result1> a qudt:QuantityValue;
16     qudt:Unit unit:DEG_C;
17   qudt:numericValue "22"^^xsd:float.
```

Listing 2.3: Complete SOSA example

Other sensor Ontologies

SAREF The Smart Appliances REFerence (SAREF) Ontology [31, 32], developed by the Netherlands Organisation for Applied Scientific Research (TNO), is another ontology with the goal of describing sensors and various interactions of IoT devices to improve interoperability.

The main concept of SAREF is the *device*. A device has a task, the description of what a device does. A function, which describes the functionality of the device which is needed to complete the task. That function consists of one or several commands, thus the command is the basic block of a function. A device can also make measurements, which is conceptually the same as an observation in SOSA.

Efforts have been made to make semantic translations from SSN to SAREF[33]. Unfortunately the translation is not 100%, which means manual intervention is required when changing from one ontology to the other.

SSN SOSA was chosen in favor of SAREF as it is a recommendation of the W3C, though the modelling of sensors and their data can both be achieved by any of those 2 ontologies. Another reason why SOSA is chosen, is that SAREF defines its own units of measurement, which are limited compared to an external ontology for units.

Web of Things (WoT) Thing Description In contrary to the above mentioned ontologies, the WoT Thing Description (TD) [34, 35] models how to interact with sensors in an interoperable way. With TD, actions can be performed directly on sensors without having to know the technologies used in communicating with the device, *the thing*, itself. This is done by using descriptions on how to use the *things*, called affordances. The TD provides metadata about different protocol bindings that are used to interact with the device, where each protocol binding is identified by URI schemes, content types and security mechanisms.

As this thesis focuses on managing data in an interoperable way, and not on interacting with sensor devices in their environment, this ontology was not considered to model sensor data.

2.1.4 Querying

Now that big datasets can be created and stored into a document, there is a need to perform actions on the dataset. In order to achieve actions like selecting a subset of triples in the dataset or creating a new dataset of those triples or updating the dataset, a query language is required.

SPARQL

A first type of query language is SPARQL [36, 37], short for SPARQL Protocol RDF Query Language, a W3C recommendation. The first version, SPARQL 1.0, only performed basic querying. The ability to update, insert and delete triples were added in SPARQL version 1.1. In order to query over a document, that document should be hosted as a SPARQL endpoint.

In practice, a server hosting datasets does not always have such an endpoint available. This results in the end user having to download the whole data set and locally query the complete data set, which then can be executed using SPARQL.

An example of a SPARQL query is shown in Listing 2.4, which queries over the SOSA example from Listing 2.3.

```
PREFIX sosa: <http://www.w3.org/ns/sosa/>
SELECT *
WHERE   { ?s ?type sosa:Observation .
          OPTIONAL { ?s ?p ?o}
        }
```

Listing 2.4: A SPARQL query on SOSA data

The *SELECT* statement indicates what values will be present in the result set of the query. The * means that the complete triples will be added to the result set. The *WHERE* clause is used to match the triples in the data set against the pattern between the curly brackets. In this example, all triples that have a predicate *http://www.w3.org/ns/sosa/Observation* match the clause. This full URL is interpreted correctly as SPARQL, just like turtle, has prefix bindings. Finally in this example the *OPTIONAL* keyword is used. Here the bindings, like the *?s* employed in the example, can be utilized to search for extra matches in the data set. The statement here would search for all the triples who have as subject

https://woutslabbinck.com/thesis/chapter2/example1.ttl#observation1.

The result of the query is a result set of the complete observation as shown in Table 2.1[3].

s	type	p	o
<observation1>	rdf:type	rdf:type	sosa:Observation
<observation1>	rdf:type	sosa:hasResult	<result1>
<observation1>	rdf:type	sosa:madeBySensor	<temperatureSensor>
<observation1>	rdf:type	sosa:resultTime	”2021-04-16T15:00:00”xsd:dateTime

Table 2.1: Result set of a SPARQL query

Comunica

Comunica [38] is a modular SPARQL query engine for the Web that allows querying over heterogeneous interfaces. In order to use the engine, no SPARQL endpoint has to be set up. As previously stated, Comunica is not limited to regular RDF formats, but it also allows querying over Linked Data Fragments (LDF) [39]. LDF was developed to provide a uniform view on all possible interfaces to RDF.

Each Linked Data set can be represented with an LDF. The LDF itself consists of three parts: Data, Metadata and Controls.

With data is meant all the triples of the fragment. Metadata are triples that describe the dataset or fragment and the controls are hypermedia links that lead to other LDFs.

A data dump can be represented as an LDF the following way: the data are the triples of the set, the metadata is then the number of triples and the file size and there are no controls.

For SPARQL query results, the data is all the triples matching the query and both the metadata and controls are empty.

As all possible ways to publish linked data can be represented by LDF, comunica is a good way of querying sets.

Another interesting thing about Comunica is that was designed to be modular, which makes it possible to define extensions.

[3]The result set in Table 2.1 contains prefixes (*sosa* and *rdf*) and a base (*<https://woutslabbinck.com/thesis/chapter2/example1.ttl#>*) to improve the readability.

2.2 Linked Data Platform

The Linked Data Platform (LDP) [2] protocol is used to host a server that can handle Create, Read, Update, and Delete (CRUD) operations for both RDF and non-RDF resources, using HTTP operations. LDP is a W3C Recommendation, created by the by the LDP Working Group.

An LDP server hosts LDP Resources (LDPRs). There are 2 kinds of LDPRs. First, there are resources whose state is fully represented by RDF, called LDP RDF Source (LDP-RS). Listing 2.3 is an example of an LDP-RS. The second kind of LDPRs are resources whose state are not represented by RDF. Those are named LDP Non-RDF Sources (LDP-NRs). Binary files like images and videos are examples of LDP-NRS. Figure 2.3 shows the distinction between those 2 types of LDPRs.

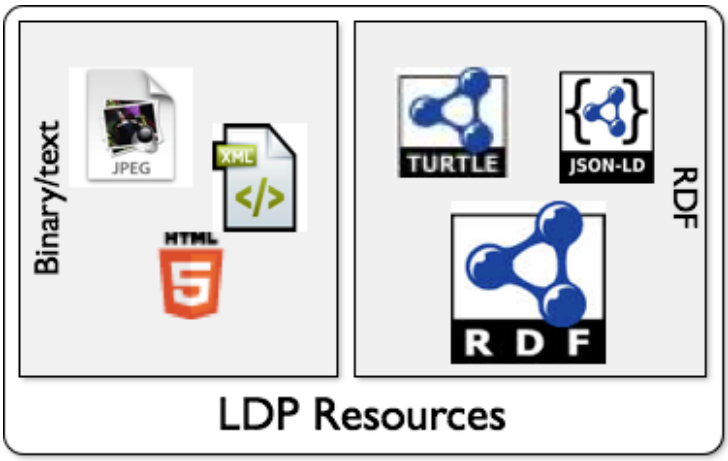

Figure 2.3: Samples of different types of LDPRs, adopted from [2]

LDP Container (LDPC) is a specialization of LDP-RS. They contain links to other LDPRs which can be manipulated using the LDP protocol. There are different kinds of LDP Containers: the Direct, the Indirect and the Basic Container LDP-BC. The Basic Container is the simplest one and only defines the links to other LDPRs. The hierarchy of LDPRs and their relationships are visualised in Figure 2.4.

Now that the different resources are explained, it can be explained how to read and write with the LDP protocol.

The document at URI https://woutslabbinck.com/thesis/chapter2/ is an LDP Basic Container. With the HTTP GET method, links to other LDPRs can be retreived. The request at Listing 2.5 results in the LDP-BC in Listing 2.6.

```
1 @prefix dcterms: <http://purl.org/dc/terms/>.
2 @prefix ldp: <http://www.w3.org/ns/ldp#>.
3
```

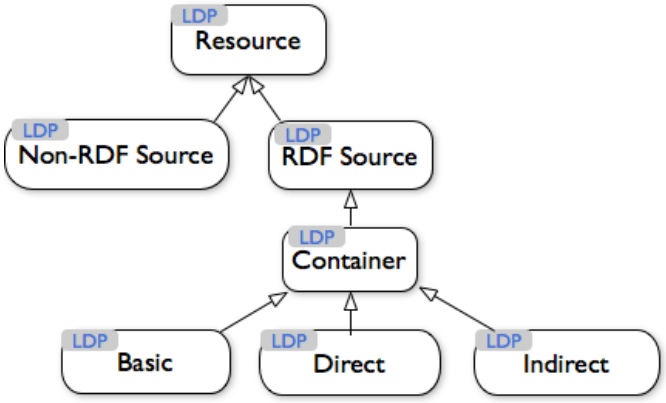

Figure 2.4: Hierarchy of the LDPRs, adopted from [2]

```
GET /thesis/chapter2/ HTTP/1.1
Host: woutslabbinck.com
Accept: text/turtle
```

Listing 2.5: HTTP GET request to an LDPC

```
4 <https://woutslabbinck.com/thesis/chapter2/> an LDP:Container, ldp:BasicContainer;
5   dcterms:title 'Data Storage of Chapter 2' .
```

Listing 2.6: LDP Basic container

Currently, this LDP-BC does not contain any resources. Writing the RDF graph of Listing 2.3 can be done with an HTTP POST method[4], which is shown in Listing 2.7. The Link header added to the HTTP request defines which type of resource has to be created and the slug defines the path that should be created.

When reading the LDP-BC at https://woutslabbinck.com/thesis/chapter2/ again with Listing 2.5, the result is now Listing 2.8. The *ldp:contains* predicate indicates that example1.ttl was indeed created. A subsequent HTTP GET request to that link would result in Listing 2.3.

```
1 @prefix dcterms: <http://purl.org/dc/terms/>.
2 @prefix ldp: <http://www.w3.org/ns/ldp#>.
3
4 <https://woutslabbinck.com/thesis/chapter2/> an LDP:Container, ldp:BasicContainer;
5   dcterms:title 'Data Storage of Chapter 2' .
```

[4]Writing of Listing 2.3 to the LDP-BC could also be achieved with an HTTP PUT method, the HTTP request would then slightly differ from Listing 2.7

```
POST /thesis/chapter2/ HTTP/1.1
Host: woutslabbinck.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Slug: example1.ttl
Content-Type: text/turtle

@base <https://woutslabbinck.com/thesis/chapter2/example1.ttl#> .
@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.

<temperatureSensor> a sosa:Sensor;
    sosa:madeObservation <observation1>.

<observation1> a sosa:Observation ;
        sosa:hasResult <result1>;
        sosa:madeBySensor <temperatureSensor>;
        sosa:resultTime "2021-04-16T15:00:00"^^xsd:dateTime.

<result1> a qudt:QuantityValue;
    qudt:Unit unit:DEG_C;
        qudt:numericValue "22"^^xsd:float.
```

Listing 2.7: HTTP POST request to an LDPC

```
6   ldp:contains <https://woutslabbinck.com/thesis/chapter2/example1.ttl> .
```

Listing 2.8: LDP Basic container, after adding example1.ttl

It is also possible to update LDPRs with an HTTP PUT/PATCH request and remove LDPRs with an HTTP DELETE request, which is explained in the LDP primer [40].

LDP thus provides an interoperable way of the CRUD operations to interact with RDF files. It allows storing the sensor data as Turtle files and updating those resources when new observations are taken.

2.3 Shapes

With RDF, a model can be created to store the sensor data and its metadata, encoded with their semantics using the SOSA ontology. However, there needs to be a guarantee that the data model is valid. This will ensure that applications that build upon that model will not run into parsing problems. Schemas or interfaces that assure such validation of data models are called shapes.

As an example, a temperature sensor makes observations. Those observations have a timestamp and a result, which in turn has a value and a measurement scale. More specifically this measurement scale has to be Celcius, Fahrenheit, or Kelvin. When there is a different kind of observation, different requirements need to be met.

For all kind of sensors, such shape has to be defined in order to guarantee interoperabillity.

2.3.1 Shape Expressions

Shape Expressions (ShEx) [41, 42] is a Domain Specific Language (DSL) used to describe and validate RDF graphs. The specification was developed by the Shape Expressions Community Group (CG) and is currently not a W3C standard.

ShEx is a grammar for defining structures. It supports cyclic data models, i.e. recursion. When validating with ShEx, the result is a shapemap which consists of both the parts that conform to the shape and parts that do not conform to the shape.

2.3.2 Shapes Constraint Language

The Shapes Constraint Language (SHACL) [43] has been developed by the W3C RDF Data Shapes Working Group with the goal of ”producing a language for defining structural constraints on RDF graphs”[44]. The working group intended for SHACL to be the unified language for all the validation approaches at that time, including ShEx. But due to core differences, ShEx and SHACL did not converge[45].

SHACL focuses on defining constraints over RDF Graphs, it does not support recursion. However, SHACL has support for SPARQL property paths.

Further in-depth comparisons of ShEx and SHACL were made in Validating RDF Data [45]. Both of the languages could have been utilized to describe the structure of the data models, although ShEx has slightly more leverage as it supports recursion. However, the main reason why ShEx was selected, is that Shape Trees, which will be explained in Section 2.4, uses ShEx as

a basis for validation.

2.3.3 ShEx Example

For this ShEx example in Listing 2.9, which describes the shape of the temperature sensor data from Listing 2.3 and further implementations, the compact, human-readable version, ShExC, will be used.

<#Sensor> is a shape which consists of 2 triple constraints. The first triple constraint has as value of the property '*a*' a value set, expressed by the brackets ([]). After the *a* predicate one of the values of that set has to be present, which in this case can only be *sosa:Sensor*. There can only be one type and it must be *sosa:Sensor* as the default cardinality in ShEx is 1. The second triple constraint holds a Shape Reference, denoted with an at symbol (@) followed by the label of the referenced shape. Here the cardinality is denoted with an asterisk (*), which means a sensor can have any number of observations and that each observation has the <#Observation> structure.

In this example there is only one notable different node constraint, more particularly a datatype. The triple constraints at line 15 and 22 both have as node constraint a datatype. This means that to validate for example an Observation, a triple with as predicate *sosa:resultTime* should be present with a correct use of *xsd:dateTime* as object.

This example also contains a cyclic reference, more specifically the <#Sensor> and <#Observation> shape reference each other.

In ShEx, base and prefixes are also present and this is written in uppercase.

ShEx has more terminology than the aforementioned example, which is unambiguously and exhaustively explained in Validating RDF Data [45]. For another high level description, the Shape Expression primer [46] can be consulted.

To validate the temperature sensor from the example, the <temperatureSensor>, which is the focus node, has to mapped to <Sensor>, the shape [5].

```
1 BASE <https://woutslabbinck.com/thesis/chapter2/shex_example.shex#>
2 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3 PREFIX sosa: <https://www.w3.org/ns/sosa/>
4 PREFIX qudt: <http://qudt.org/schema/qudt/>
5 PREFIX unit: <http://qudt.org/vocab/unit/>
6
```

[5] With the Simple Online Validator tool [47], this validation can be verified when putting the ShEx example left, the RDF example right and <temperatureSensor>@<Sensor> in the Query Map.

```
<Sensor> {
    a [sosa:Sensor] ;
    sosa:madeObservation @<Observation> *
}

<Observation> {
    a [sosa:Observation] ;
    sosa:madeBySensor @<Sensor> ;
    sosa:resultTime xsd:dateTime ;
    sosa:hasResult @<Result>
}

<Result> {
    a [qudt:QuantityValue] ;
    qudt:Unit [unit:DEG_C] ;
    qudt:numericValue xsd:float
}
```

Listing 2.9: Shape Expression for a temperature sensor in shexC

2.4 Shape Trees

Sensor data can be transformed into RDF data with the SOSA ontology, the created RDF graphs can be stored in an LDP and it is possible to validate them using ShEx. Unfortunately, the LDP does not ensure that the stored data actually conforms with the shape as LDP just provides the methods to write and read data to a platform. A file should be verified before it is stored to a LDP. It is possible to add a process in the pipeline right before storing the data to validate that the data conforms to the shape. Though, services that read the data can never be sure that such a process was put in place, resulting in defensive programming to verify the correctness of that data.

Validation before storing is exactly what Shape Trees [9] does. It was developed by the W3C Solid Community Group as the foundation for the Solid Application Interoperability Work Item [48].

When Shape Trees are used to validate, they become managed resources and containers, as described in Section 2.2.

The operations to add support to an LDP with Shape Trees are explained next [6].

[6]The requests and syntax that are used for the Shape Trees operations in this section work with the implementation created by Eric Prud'hommeaux on from the shapetrees github repository [49].

The first step is to indicate that a container only contains validated resources. This is done by the ”plant” operation. After the plant operation, the container becomes a managed container. Listing 2.10 is a Shape Tree for sensor RDF data. When planted, the managed container *<#container>* can only contain*<#sensor>* resources that are validated by the ShEx file, defined in Listing 2.9.

```
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix st: <http://www.w3.org/ns/shapetree#> .

<#container> st:expectsType ldp:Container ;
  st:contains <#sensor> .

<#sensor> st:expectsType ldp:Resource ;
  st:validatedBy <Sensor.shex#Sensor> .
```

Listing 2.10: Shape Tree in Turtle format

To initiate a Shape Tree, the plant operation, an HTTP POST or PUT request is sent to a platform which supports Shape Trees; a working platform can be found on the shapetrees github repository [50]. In this request a Link header is required which represents a Shape Tree together with a an additional Link header for creating the LDPR, which according to that Shape Tree has to be an LDP Container. With a POST request, a slug is also needed to indicate the filename of the resource that has to be created. Listing 2.11 is a plant request to the local hosted test-suite Shape Tree [50].

```
POST /Data/ HTTP/1.1
Host: localhost:8080
Content-Type: text/turtle
Link: <http://www.w3.org/ns/ldp#Container>; rel="type",
 ↪  <https://woutslabbinck.com/thesis/chapter2/ShapeTree.ttl#container>;
 ↪  rel="shapeTree"
Slug: Sensors

PREFIX ldp: <http://www.w3.org/ns/ldp#>
[] ldp:app <http://store.example/> .
<http://store.example/> ldp:name "Collection&ShapeTree" .
```

Listing 2.11: Shape Trees Plant operation to a local hosted LDP that supports Shape Trees

To create a data instance two Link headers are needed: one to indicate the root of the graph that will be validated against the shape file, the other to indicate the type of resource that will be created. Additionally, a slug is added to the POST request to indicate the filename of the resource. The Create Data Instance request is shown in Listing 2.12. As it is demonstrated

in Section 2.3, this body is already validated and will thus be created. However, when any important triple was missing, no resource would be created and an HTTP Unprocessable Entity error, code 422, would be returned.

```
POST /Data/Sensors/ HTTP/1.1
Host: localhost:8080
Content-Type: text/turtle
Link: <http://www.w3.org/ns/ldp#Resource>;rel="type",
↪  <temperatureSensor>;rel="root"
Slug: temperatureSensor.ttl

@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.

<temperatureSensor> a sosa:Sensor;
    sosa:madeObservation <observation1>.

<observation1> a sosa:Observation ;
        sosa:hasResult <result1>;
        sosa:madeBySensor <temperatureSensor>;
        sosa:resultTime "2021-04-16T15:00:00"^^xsd:dateTime.

<result1> a qudt:QuantityValue;
    qudt:Unit unit:DEG_C;
        qudt:numericValue "22"^^xsd:float.
```

Listing 2.12: Shape Trees Create Data Instance operation

At the end of those 2 operations, the following resources hierarchy is present in the Shape Tree supported LDP.

Obviously, Shape Tree methods exist to update and delete instances as well as a method to unplant the Shape Tree.

2.5 Social Linked Data

There are several requirements if an end user aspires to have full control over his/her data. First a mechanism that handles authentication is needed to ensure that end users can be identified. Additionally, when the end user wants to share data with a certain individual, it must be possible for the end user to choose what type of access control is granted. When the end user wants to share information, that type is probably read only. For collaborating however, both read and write access should be granted. An authorization service that can handle access control over the resources and is thus essential.

The Social Linked Data (Solid) Protocol [7] is a solution to the needs listed above. The protocol is under development by the W3C Solid Community Group with the high level goal of giving people control over their online data. Solid was designed as a decentralized platform for social applications [51]. Applications build for the Solid ecosystem work with data that they do not own themselves as the data resides in personal online datastores (pods) of the end user. Applications can directly read data from and write data to these pods. This approach is in contrast to the current social media platforms, as they are gathering data through a Walled Garden approach. A Walled Garden approach is "a closed ecosystem in which all the operations are controlled by the ecosystem operator" [52].

The Solid Protocol does not re-invent everything from scratch, but rather bases their protocols on existing W3C Recommendations.

Solid provides storage capabilities based on mechanisms corresponding to LDP, which was already explained in Section 2.2.

For authentication the Web Identity and Discovery (WebID) specification [8] is used in combination with the Solid-OIDC [7] [53] specification .

Authorisation and access control are handled by Web Access Control (WAC) [54]. In Solid, due to WAC, each resource has an additional set of statements about authorisation indicating who has access to the resource and what type of access they have. The statements are stored in a separate file, called an Access Control List (ACL) Resource, which are built with the Access Control Ontology [55].

[7] OIDC is short for OpenID Connect, which is a simple identity layer on top of the OAuth 2 protocol.

2.6 Fragmentation strategy

A single observation and the related metadata does not require much space. However, a sensor does not stop observing. As the data volume grows, PATCH requests on a single RDF file will gradually take more time, since the delay of operations increases linearly with the growth of data volume. [56].

Another downside of having one RDF file as storage, is that it becomes hard to query the file. When interested in reading the last measurement, the whole file needs to be retrieved with a GET request before it can be queried to retrieve that measurement. The file has become a data dump.

Therefore, there is a need to fragment the data in a smart way, which both benefits writing to a platform and querying the whole dataset.

2.6.1 TREE hypermedia

An approach to fragmenting data can be achieved with TREE Hypermedia [10]. The idea is to have an alternative to pagination. In pagination, a collection is fragmented and it is possible to navigate to the next or previous page. In TREE the collection of items persists, but now navigating is done based on relations which have a link to other fragments of the collection. A TREE fragment consists of a collection and a node. A collection has members which can be found in the fragment itself, a node can be seen as page and can have links to other TREE fragments. There is one special node per fragmentation, the root node. From the root node, the whole collection with all the members can be retrieved. If a collection has a *tree:view* predicate to a node, than that node is a root node for that collection.

A relation can be more specific than a link to another page. It is possible to add properties that give some information about the items in the collection when following that link. An example of a specialized relation is the possibility to compare timestamps, which can be very useful when narrowing down a query.

An example of a collection with a root node is given in Listing 2.13. The root node has a *GreaterThanOrEqualToRelation* on the *resultTime* of the observation, which indicates that all observations will be more recent than the timestamp denoted in *tree:value*. Traversing the link of the relation will result in Listing 2.14. The collection of that fragment consists of 1 observation.

```
1 @prefix tree: <https://w3id.org/tree#>.
2 @prefix ldp: <http://www.w3.org/ns/ldp#>.
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
4
```

```
<root.ttl#Collection> a tree:Collection;
    tree:view <root.ttl> .

<root.ttl> a tree:Node;
    tree:relation [
        a tree:GreaterThanOrEqualToRelation ;
        tree:path sosa:resultTime ;
        tree:node <observations.ttl> ;
        tree:value "2021-04-09T00:00:00Z"^^xsd:dateTime
    ] .
```

Listing 2.13: A collection of observations with a root node of a TREE fragment

```
@prefix tree: <https://w3id.org/tree#>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.

<observations.ttl#Collection> a tree:Collection;
    tree:member <observation1> .

<observations.ttl> a tree:Node .

<observation1> a sosa:Observation ;
  sosa:hasResult <result1>;
  sosa:madeBySensor <temperatureSensor>;
  sosa:resultTime "2021-04-16T15:00:00"^^xsd:dateTime.

<result1> a qudt:QuantityValue;
    qudt:Unit unit:DEG_C;
  qudt:numericValue "22"^^xsd:float.
```

Listing 2.14: A collection of observations of a TREE fragment

Currently, there is an implementation of a typeahead search interface [57] based on prefix relation. The collection that the interface queries consists of thousands of fragmented files, which in total are approximately 350 MB in size.

2.6.2 Linked Data Event Streams

The fragmentation strategy allows storing a substantial amount of items. Though, sometimes it is not interesting to keep all that information. There is a need for a mechanism that decides

which items to keep and which to drop from the collection. This description fits a retention policy.

Linked Data Even Streams (LDES) [11], an extension to TREE Hypermedia, provides an affordance to define such a retention policy. In LDES, the TREE collection is immutable. This indicates that once an item is added to the collection, the use of LDES guarantees that the item will stay there. The guarantee to keep all data is the opposite of what a retention policy accomplishes, which is why LDES allows defining how data will be removed from the collection when certain conditions are met.

Recall the node from TREE, this is extended in LDES with *lides:retentionPolicy*. There are two types declared: time-based and version-based retention policies. The time-based retention policy defines the minimal amount of time an item must be stored. The version-based indicates the minimal number of items to be stored. A combination of both is also possible.

It is sufficient for a collection to make the root collection an *lides:eventStream*, which is an extension of the *tree:Collection* class. An example of an LDES collection can be seen in Listing 2.15. Note that this example is the same collection as in Listing 2.13. However, since it is a *lides:eventStream* it is guaranteed that all members are immutable. The *lides:LatestVersionSubset* retention policy indicates that only the 100 most recent members must be stored from a subset of the collection. The *100* comes from *lides:amount* and to calculate the most recent members a *tree:path* is added with a SHACL path to retrieve a timestamp of the members. An extra predicate, *lides:versionKey*, may be added to the version-based policy to specify a subgroup of the collection on which the policy applies. When not added, which is the case in the example, the policy applies on the whole collection.

```
1 @prefix tree: <https://w3id.org/tree#>.
2 @prefix ldp: <http://www.w3.org/ns/ldp#>.
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
4 @prefix ldes: <https://w3id.org/ldes#>.
5
6 <root.ttl#Collection> a ldes:EventStream ;
7     tree:view <root.ttl> .
8
9 <root.ttl> a tree:Node;
10     ldes:retentionPolicy <retentionPolicy>;
11     tree:relation [
12         a tree:GreaterThanOrEqualToRelation ;
13         tree:path sosa:resultTime ;
14         tree:node <observations.ttl> ;
15         tree:value "2021-04-09T00:00:00Z"^^xsd:dateTime
16     ] .
17
```

```
18 <retentionPolicy> a ldes:LatestVersionSubset;
19     tree:path sosa:resultTime ;
20     ldes:amount 100 .
```

Listing 2.15: A LDES collection of observations with a retention policy

2.7 RDF Mapping Language

Mapping observations from sensors can be done with SOSA, as explained in Section 2.1.3, where a handcrafted example was presented. However, this required the intervention of a person who could translate the generated sensor data to the RDF representation. This process is a complex and time consuming task. Moreover, humans often make mistakes [58]. Consequently, manually transforming the data is not scalable, which indicates that a translation service is required.

The RDF Mapping Language (RML) [59] is an affordance to this translation problem. RML was developed by the University of Ghent and is a superset of the W3C recommendation RDB toRDF Mapping Language (R2RML) [60]. R2RML provides functionality to map relational databases to RDF. With RML it is also possible to map from other structured formats, such as JSON and CSV, to RDF.

2.8 Sensor network and communication

In the previous sections, there was a focus on semantics and services built on top of data. This data is generated by sensors that come in an IoT solution. In order for this data to reach these services, communication is required. Often, smart devices are placed at remote locations, which means that a wired connection is not feasible. Another consideration is that a smart device should last long and again, due to the remote locations, the devices are generally battery powered, which implies the total amount of power is limited. For this reason, a long range low-powered communication technology is required. A survey from Raza et al. [61] showed that many standards exist, such as DASH-7, IEEE 802.15.4, NarrowBand IoT (NB-IoT), Long Range Wide Area Network (LoRaWAN), ... This thesis uses a LoRaWAN setup, due to its long range and openness [62].

A LoRaWAN uses a star of stars topology, which can be seen in Figure 2.5. The IoT devices are the end nodes, which transmit packets to one or multiple gateways via the LoRa physical layer. Each gateway is then connected to the Network Server (NS). Applications and services then have a connection to the NS to retrieve the data generated by the IoT devices.

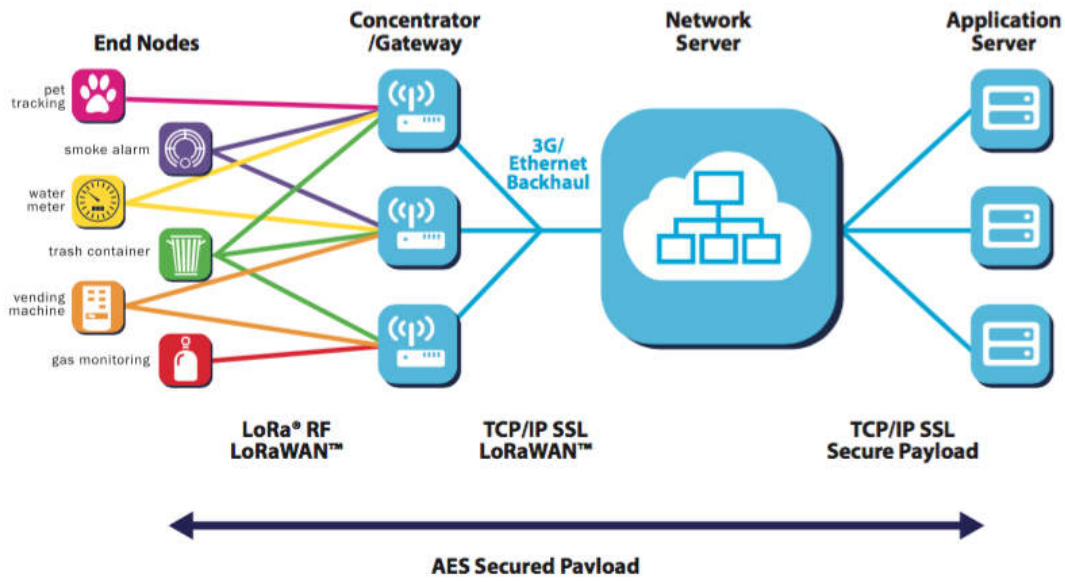

Figure 2.5: LoRaWAN architecture: star of stars topology, adopted form [3]

2.9 Broker technology

The publish-subscribe pattern is an application layer protocol to bring the data messages from the sensor to services in a modular way. Neither the service responsible for processing, validating and storing data, nor the sensor that generates the data, do not need to know each other. The only thing that has to be known beforehand is the topic, where the unique identifier for the sensor can be used, and the application, called a message broker, itself. Therefore the NS publishes data to the broker with a specific topic. When a service is subscribed to that specific topic, the broker sends that data to the service. As a message broker, the Message Queuing Telemetry Transport (MQTT) protocol [63] was chosen. MQTT is a lightweight protocol, suitable for routing resources for low power devices and is already used in many IoT applications [64]. An implementation of this protocol is provided by Mosquitto [65, 66], developed by Eclipse.

2.10 Challenges

2.10.1 RDF validation

Shapes for RDF were created to guarantee that a certain graph in an RDF resource follows a specific model. This means that no custom SPARQL queries should be written to validate the expectations that certain elements were indeed in the RDF resource [45].

Nevertheless, the task to manually validate every graph that is used in a program, even with shapes, is quite extensive. For this reason, the Shape Tree specification [9] was created to validate graphs for LDPs.

No research has yet been done to verify if validation has an impact on the performance of an LDP.

2.10.2 PATCH requests

An implementation for adding sensor data to a Solid Pod was already presented in [56]. However, the author noticed that as files grew in data volume, the delay of PATCH requests increased linearly with growth in data volume.

A proposed strategy to solve the slow PATCH requests, is by guaranteeing that the stored resource remains small, which is done by fragmenting the data using the TREE Hypermedia specification 2.6.1.

3

Architecture

An architecture was designed in order to develop the proposed solution. In this prototype, sensor data is stored on a Solid Pod, indicating that owner has full control over the data. A high level overview is given in Figure 3.1. It consists of several sensors that generate data. This data is transferred via several gateways to a service which transforms the data to Linked Data. This service also stores the data to an LDP which can validate the data. Finally, when the data is validated, the data is stored on a Solid Pod.

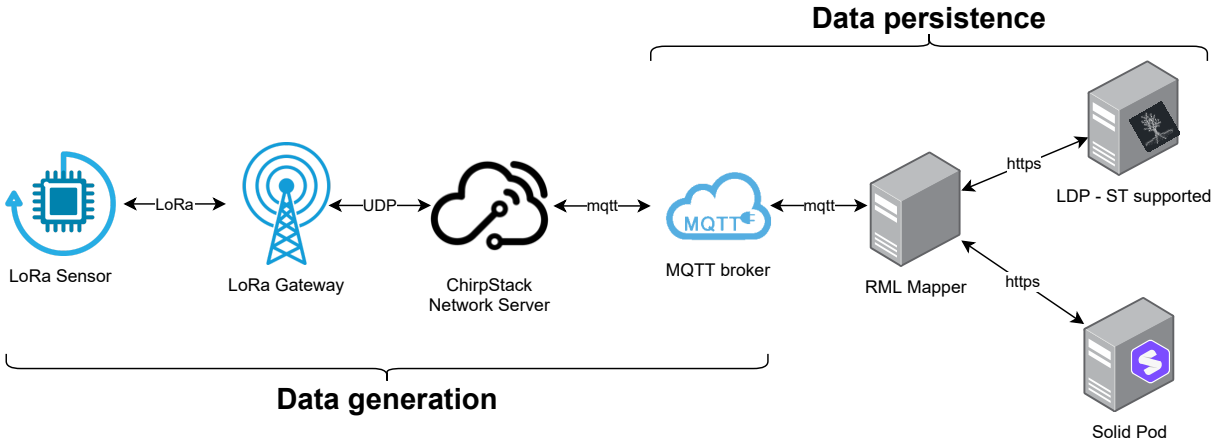

Figure 3.1: Architecture

A more detailed explanation is given in the following sections.

3.1 Data generation

The sensor devices measure different properties in their environment, such as temperature, humidity, pressure, ... Every ten minutes, a packet containing these measurements is transmitted to a LoRa Gateway. These messages have a proprietary, compact format. This allows the transmission time of the sensor to be as small as possible, as the transmission is a costly operation in terms of power consumption. The LoRa Gateway is configured to send the data from the registered sensors to the ChirpStack Gateway Bridge, a component of the ChirpStack project [67]. The ChirpStack project is an open-source solution to manage LoRaWAN networks. The ChirpStack Gateway Bridge publishes the sensor data to the publish/subscribe MQTT broker with a specific topic, which is reserved communicating with the ChirpStack Network Server (NS). The ChirpStack NS subscribes to that topic and is notified when there is a new message. The NS de-duplicates these messages and publishes them to the MQTT broker under a specific topic, known to both the NS and the Mapper service, to which the RML Mapper is subscribed. This de-duplicating is marked on the UML sequence diagram in Figure 3.2 with an 'Opt' frame as only the first message will be published to the broker for the RML Mapper.

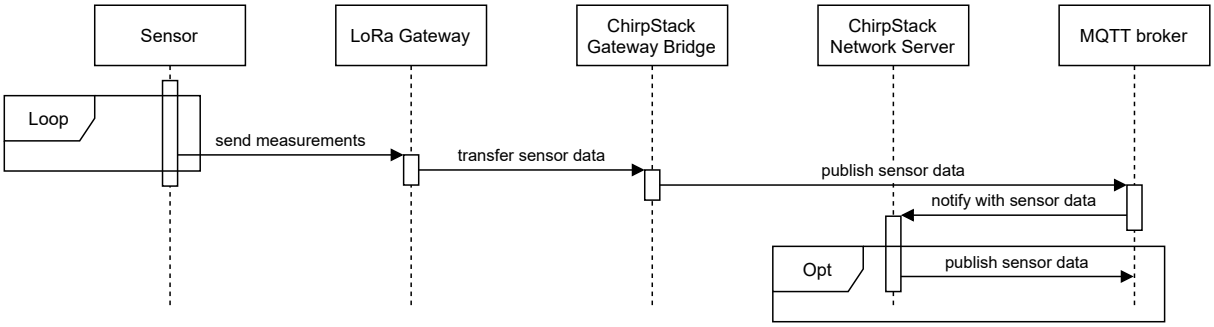

Figure 3.2: Sequence diagram of the flow from sensor to the MQTT broker

3.2 Data persistence

When an event is received from the MQTT broker, the RML Mapper decodes the sensor data to an intermediary representation. The service then transforms the sensor data to Linked Data and transmits it over HTTPS to the LDP server, which has Shape Trees support. Then, when this data is validated and thus stored in the LDP, the RML Mapper finally puts the Linked Data in a Solid Pod, hosted by a Node Solid Server (NSS) [68]. These different steps are visualized in Figure 3.3. The 'Opt' frame indicates that storing to the Solid Pod only occurs when the

validation of the sensor data was successful.

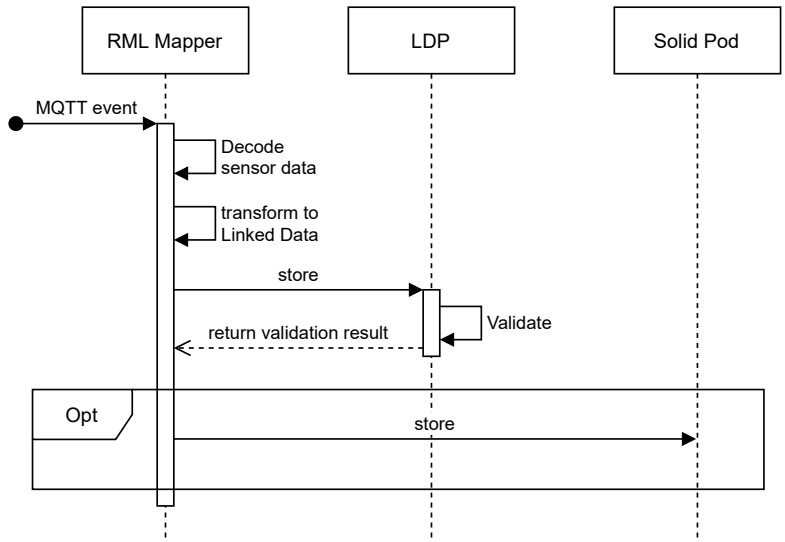

Figure 3.3: Sequence diagram of storing the sensor data to the Solid Pod

3.3 Fragmentation and retention service

Over time, the data volume grows on the LDP. The result is that when no measures are taken, adding additional data becomes slow. The RML Mapper counteracts the performance drop by performing two extra services. The first one is the fragmentation service. This service prevents single RDF files from becoming too large through fragmenting the whole data set into multiple smaller files. The second service is responsible for data retention. A retention policy is defined to limit the amount of measurements stored on the LDP.

As a result, the transformed sensor data stored in the LDP is a TREE collection that uses the SOSA Ontology as a model.

In Section 2.1.3, an example of a *sosa:Sensor* was given which was capable of sensing one property. However, a device that senses multiple properties, can be grouped under the *sosa:platform* entity. Thus, when a sensor is capable of measuring a multitude of properties, the sensor acts like a *sosa:platform* with a sensing ability; a *sosa:sensor* for each property. Hence, all the information about the sensor is stored together in one file on the LDP as *platform_{uuid}.ttl*. Where the uuid is the unique universal identifier of the sensor.

Recall Section 2.6.1, where it was explained that a collection consists of different TREE fragments where each fragment consists of a node and a collection. The most important node, the root node, has relation(s) to all TREE fragments, indicating that the whole TREE can thus be reached

from the root. When the first message of a platform reaches the RML Mapper, together with the platform file, both the root node and a TREE fragment are created. All further messages with new observations are then appended to the TREE fragment. Figure 3.4 shows the file structure of how the sensor data is stored and validated on the LDP. The root node is stored in the file *root_{uuid}.ttl* and the TREE fragment with all the observations as member of the collection in *today_{uuid}.ttl*.

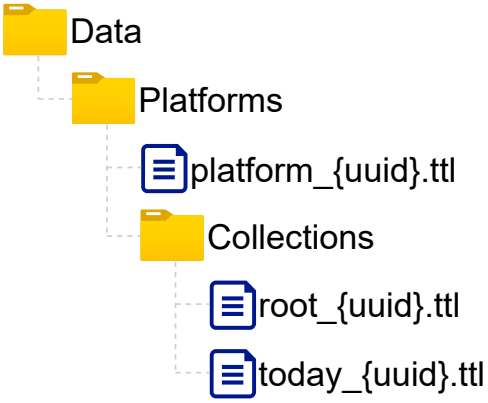

Figure 3.4: Storage structure on the LDP

Storing all observations in one file, however, is not a scalable approach. Furthermore, it would then not take advantage of the mechanisms of the TREE Hypermedia specification. The solution to this problem is defined by a time-based fragmentation strategy, where the observations are stored as shown in Figure 3.5. The core idea behind this time-based approach is that recent observations are most interesting to the end user and thus querying them should deliver results fast. Another consideration is the fact that the *today* file will have frequent changes, which means it is favorable to keep the size small allowing to provide rapid updates.

Consequently, *today_{uuid}.ttl* only contains the observations of the day itself. Correspondingly, *last_7_days_{uuid}.ttl* contain observations of the given week and *current_month_{uuid}.ttl* of this month. Finally, the older observations are all placed together per month with file names following the scheme *{year}_{month}_{uuid}.ttl*.

In order to keep these growing sizes manageable, retention policies are defined with LDES, which are executed later. Hence, next to decoding, transforming and storing the data to LDPs, the RML Mapper is also responsible for fragmenting and executing the retention policy as can be seen in Figure 3.6. It is designed to perform these services daily at midnight, nevertheless, it is modular in the sense that when a sensor measures more frequently it can be executed more often and vice versa.

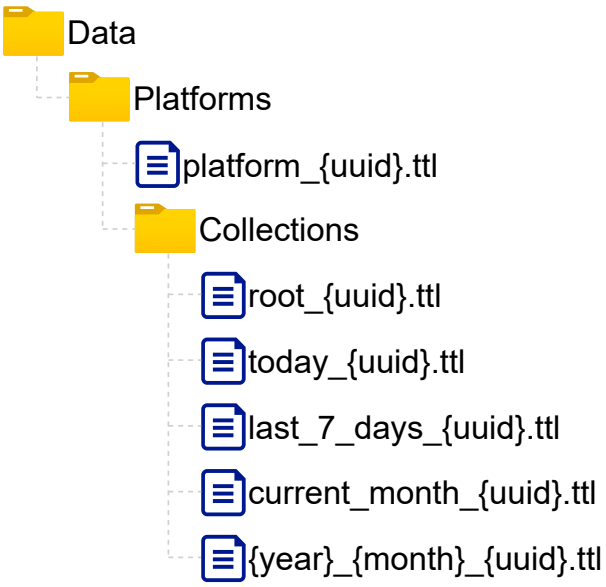

Figure 3.5: Fragmentation storage structure on the LDP

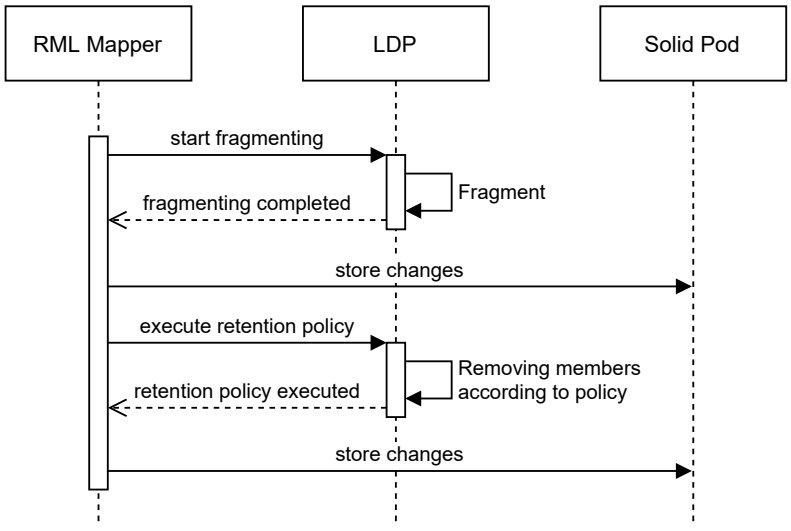

Figure 3.6: Sequence diagram of the fragmentation service and the retention policy execution

4

Implementation

In this chapter, an elaboration is given about how the mechanisms for data persistence, fragmentation and retention from Chapter 3 were implemented in software.

4.1 Converting sensor data to Linked Data

As described in Section 3.1, the sensor data is received by a number of gateways before being published to the MQTT broker. The RML Mapper is subscribed to the topic of the sensor, delivering data in the proprietary format.

The first step for this service, consists of decoding the message and storing it in an intermediate representation, more specifically the JSON format. After this translation, the data is enriched with units as described in the QUDT ontology [28] and the name of what is measured. For example, for a temperature measurement the QUDT unit *unit:DEG_C* is added together with the property name "Air temperature". Finally, using RML [59], the intermediate representation is translated to Linked Data with SOSA as a model.

In Appendix B, an example of an incoming data packet is given. The result of decoding and enriching this packet to its intermediate representation can be seen in Appendix C. When this intermediate representation is mapped with the RML file from Appendix F the output Turtle

file shown in Appendix D.

4.2 Adding new data to the LDP

When the data is represented with the SOSA model, there are two implementations which POST sensor data to an LDP. The first one adds the sensor data, represented by the SOSA model, to one file on the LDP. Since this solution is not scalable due to PATCH requests to the LDP taking too long, a second approach tries to solve this scalability problem by introducing a fragmentation strategy. Through this strategy, a PATCH request for new measurements will not take that much time anymore, because the size of the stored data where the PATCH request is executed remains small. It has been shown [56] that for small sizes PATCH requests demonstrate adequate performance.

For both approaches, a first step consists of planting the Shape Tree. The Shape Tree file must be hosted by the LDP and its URL must be known. In this file, links to shapes (in the form of an URL, normally with a HTTP fragment) are present as subject in *st:validatedBy* predicates. The content of these shapes defines the structure of the model.

An algorithm which checks whether the given Shape Tree was planted is given in 1. When the Shape Tree has not been planted yet, the Plant Shape Tree operation issues a HTTP POST request as described in Listing 2.11.

Algorithm 1: Planting a Shape Tree

Input: URL of the Shape Tree, of the LDP and the name of the root

Output: rootURL where Data Instances can be created

```
shapeTreeURL                          // URL where the Shape Tree file is hosted
Function PlantShapeTree(LDPURL, shapeTreeURL, rootContainerName):
    planted = send HTTP request to LDPURL and check if rootContainerName exists
    if not planted then
        rootURL = plant Shape Tree using LDPURL, shapeTreeURL and rootContainerName
    else
        rootURL = LDPURL+"/"+rootContainerName+"/"
    return rootURL
```

The ShEx file containing all the shapes for both the implementations can be seen in Appendix G. Appendix H contains 2 Shape Trees, the first implementation uses the first Shape Tree and the second implementation the second Shape Tree.

4.2.1 Adding SOSA sensor data to an LDP with Shape Tree Support

The first approach for storing and validating sensor data to an LDP, is by adding measurements directly to its dedicated file on the LDP.

The algorithm for adding measurements to such an LDP is shown in Algorithm 2. Next to the sensor data, there are two key input parameters required. The first argument is the URL of the LDP, the platform where the data resides. The second argument is the URL of a hosted Shape Tree file, which is needed for the first step: executing Algorithm 1, which may perform a plant operation.

Next, a check is performed whether the current platform of the sensorData already exists. If there is no such platform, this is the first time the function is called. The sensorData is then directly added to the LDP with the HTTP POST request to the LDP. Only when the data is validated with the ShEx file, creating the data instance will succeed. However, when the platform does exist already, the sensorData is added via a PATCH request. Patching a data instance is done using SPARQL Update queries without a WHERE clause. An example of a patch request is given in Listing 4.1, which adds a second observation to a sensor modelled with SOSA, see Listing 2.3.

```
PATCH /thesis/chapter2/example1 HTTP/1.1
Host: woutslabbinck.com
Content-Type: application/sparql-query

INSERT DATA {
<#temperatureSensor>  <http://www.w3.org/ns/sosa/madeObservation> <#observation2> .
<#observation2> a <http://www.w3.org/ns/sosa/Observation> .
<#observation2> <http://www.w3.org/ns/sosa/hasResult> <#result2> .
<#observation2> <http://www.w3.org/ns/sosa/madeBySensor> <#temperatureSensor> .
<#observation2> <http://www.w3.org/ns/sosa/resultTime>
"2020-08-25T07:05:32Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<#result2> a  <http://qudt.org/schema/qudt/QuantityValue>.
<#result2> <http://qudt.org/schema/qudt/Unit> <http://qudt.org/vocab/unit/DEG_C> .
<#result2> <http://qudt.org/schema/qudt/numericValue>
"22"^^<http://www.w3.org/2001/XMLSchema#float> .
}
```

Listing 4.1: HTTP PATCH request to the LDP

Algorithm 2: Storing SOSA data to a shape tree supported LDP

Input: Sensor data in SOSA format
Output: HTTP message and status code of result
sensorData = sensor data translated to SOSA
LDPURL `// URL of the Linked Data Platform`
shapeTreeURL `// URL where the Shape Tree file is hosted`
Function `StoreDataToLDP(`*sensorData, LDPURL, shapeTreeURL*`):`
    rootURL = PlantShapeTree(LDPURL, shapeTreeURL, "platform") `// see Algorithm 1`
    platformName = unique platform name extracted from sensorData
    uniquePlatformInstance = HTTP GET rootURL+platformName
    if *uniquePlatformInstance does not exist* **then**
        response = Create Data Instance using sensorData, rootURL and platformName
    else
        response = Patch Data Instance using sensorData, rootURL and platformName
    return response

4.2.2 Adding SOSA sensor data as part of a TREE collection to an LDP with Shape Tree support

The second approach for storing the sensor data is a first step towards creating the storage structure of the proposed solution. Here, the SOSA data is first transformed to a TREE collection. Algorithm 3 shows how the transformation is done and how the data is stored to the LDP. When the platform and the root were already posted, data will only be added to one file. This collection will keep on growing during the day, but will be cleared every day since the fragmentation policy is in use. This, in contrast to the first approach in previous section, results in a scalable solution for storing a collection of sensor data in an LDP.

4.3 Fragmentation

In the LDP, new observations are added to the collection, located in a single RDF file. This implies that adding to or querying the data is still a slow process. A fragmentation strategy was designed and implemented to distribute the collection over multiple files, enhancing the velocity of manipulating the data.

The fragmentation service has two functions. One function updates the timestamps in the relations of the root node based on the current time. The second function moves members from

one TREE fragment to the other based on the timestamps in the relation, for all relations.

Updating the root node is explained in Algorithm 4. There, the timestamps in the relations to the *today* file, *last_7_days* file and *current_month* file are updated. At the beginning of a new month, a new archive month relation is added.

At the start of a month, the relations for the last 7 days and current month are removed from the root node. That is because after fragmentation, their TREE fragments are empty. For example, when the timestamps represents February first, all the elements that could reside in the *last_7_days* file or the *current_month* file, are placed in the archive month. This means those two files are empty and a relation to them from the root node would not make sense. Therefore, they are removed from the root node.

As mentioned in the beginning of this section, the actual fragmentation is executed in the second function. This function is presented in pseudocode in Algorithm 5. The move operations are in fact two PATCH operations. The first PATCH operation deletes the members from the TREE fragment 'from relation' whereas the second PATCH operation adds the members to the TREE fragment 'to relation'.

These two functions make it possible to clear the *today* file every day. Consequently, the addition of new observations to the collection, using Algorithm 3, remains efficient, even when the size of the collection grows big.

4.4 Retention policy

In this section, the extraction of members from a collection according to the retention policy will be explained first. Extraction is the operation that retrieves a list of members that should no longer be in a the collection; a so called *"RemoveList"*. Second, an explanation will be given on how the members are actually removed from the TREE fragments.

During the thesis, several suggestions were made to Pieter Colpaert, the editor of the specification. They were also implemented to demonstrate their usefulness. The most important being a change to the *ldes:versionKey* predicate. Originally, the object of this triple was a SHACL path. This implied that a subset can be defined which has the *sosa:ObservableProperty* predicate. Using a SHACL path, however, did not allow an end user to define that temperature should be observed. In Section 4.4, the adapted version and how it works is given.

Recall from Section 2.6.2 that in the LDES specification [11], there are two kind of retention policies defined: a time-based and a version-based retention policy.

The *ldes:DurationAgoPolicy* is the time-based retention policy. Time-based means that when a duration is given to this policy, all the members of the collection which are older than the current time minus this duration are no longer guaranteed by the policy to be kept. This policy has two predicates. The first one, indicated by *tree:value*, is the duration. The object of this triple is a literal of the type *xsd:duration*. The second predicate is the SHACL path, indicated by *tree:path*. This SHACL path is used to retrieve the timestamps from the *tree:member*s. In Listing 4.2, an example of a *ldes:DurationAgoPolicy* can be seen, which assures that all members from a year ago until now will not be deleted. All the other members of the collection however may be removed.

```
1 <p1> a ldes:DurationAgoPolicy ;
2      tree:path sosa:resultTime ;
3      tree:value "P1Y"^^xsd:duration .
```

Listing 4.2: An example of a time-based retention policy

The *ldes:LatestVersionSubset* is the version-based retention policy, where the emphasis is on the number of members that must be kept. An *ldes:amount* predicate is used to specify the number of members that must be kept in the collection. Just like the time-based policy, the version-based policy has a *tree:path* predicate with the same functionality. The most interesting predicate is the *ldes:versionKey*, which is used to define the subset. A *ldes:VersionKey* consists of a *tree:path* and an optional *tree:value*. A member is in the subset on which the policy applies when the member contains the path. When the value is present, the object from the path must match the object of *tree:value*. In Listing 4.3, an example of an *ldes:LatestVersionSubset* can be seen, which assures that at least two members will not be deleted. The two members that will remain for sure, are the most recent ones. Just like in the time-based retention policy, all the other members may be removed from the collection.

```
1 <p2> a ldes:LatestVersionSubset;
2      tree:path sosa:resultTime ;
3      ldes:amount 2 ;
4      ldes:versionKey <vk1>.
5
6 <vk1> a ldes:VersionKey;
7     tree:path sosa:observedProperty ;
8     tree:value <http://localhost:8080/shapes/ObservableProperties.ttlwindSpeed>.
```

Listing 4.3: An example of the of a version-based retention policy

Single retention policies are easily interpreted and executed by an algorithm. However, with LDES it is possible to combine different policies all together. When a single policy is interpreted, a list can be received containing all the members which must remain. And in the case of a version-based policy a list that contains all the members that were not in the version subset is

retrieved: the untouched list. When taking the difference of the whole collection with the union of the remain list and the untouched list[1], the result is the list containing the members which should be removed.

Combining policies means complying with all the retention policies[2]. In extreme situations, when a single policy concludes that a member of a collection must remain, then that member must remain, even when all the other policies declare that this element can be removed.

Equation 4.4 visualizes this mathematically. The elements that must remain in the collection consist of the union of the remain lists together with the intersection of all the untouched lists. The list of elements that can be removed (Π) from the collection can be calculated by taking the difference between the collection (Ω) and the remain list (K). Algorithm 6 can be used to retrieve the list of members which should be removed when multiple policies are present. The algorithm describes how to extract the remain and untouched list from both policies. Furthermore, it explains how the remove list of all the policies is calculated. The function *ExtractMembers* is a modular function, indicating that when more policies are designed, they could be added in this function.

$$
\begin{aligned}
K &= \kappa_1 \cup \kappa_2 \cup \cdots \cup \kappa_n && K \text{ represents the members kept} \\
\Upsilon &= \upsilon_1 \cap \upsilon_n \cap \cdots \cap \upsilon_n && \Upsilon \text{ represents the members not in the version subset (untouched)} \\
\Pi &= \Omega \setminus \big(K \cup \Upsilon\big) && \Pi \text{ represents the members to remove}
\end{aligned}
$$

where:

κ_i Remain list i containing the members that must be kept according to policy i

υ_i Untouched list i containing the members that are left untouched according to policy i

Ω All the members of the collection

Equation 4.1: Mathematical description of multiple policies

When the retention policy is interpreted by Algorithm 6, it is clear which members must be kept and which ones should be removed.

Algorithm 7 executes the whole retention policy for a collection. First the fragmented collection is put together and the policies are extracted using the root. Then the members who should

[1]The untouched list in the case of the time-based policy is always empty.

[2]Note that it is not interesting to combine multiple time-based policies as they result in only the strongest policy, which is the one with the biggest duration value, being effectively executed.

be removed are calculated for the whole collection. This remove list is then used to extract the members per file that should be removed. In the final step each file is patched with the remove per file list.

4.5 Copying to a Solid Pod

Recall from the Section 3.2 and the flow when a new message arrives from Figure 3.2 that sensor data is only added to the Solid Pod when the data is validated due to Shape Trees. This implies that when Algorithm 2 or 3 succeeds, a similar algorithm to those two is executed with two differences. The first difference is that the service executing the algorithm must have valid credentials for the given Solid Pod and authenticate with them. The second difference is that no Shape Tree is planted. As input, the LDP URL used for this service, is the URL of the Solid Pod (which is an LDP, see Section 2.5).

The step of copying the data from the LDP to the NSS could be avoided, when the LDP of the NSS itself could support Shape Trees. An option was considered to implement this functionality, but the integration of Shape Trees in Solid would be implemented by the developers of Shape Trees. This insight lead to the decision of not implementing the integration, but instead using the intermediary Shape Tree supported LDP.

4.6 Editing the access control list for a collection

Solid uses the WAC/ACL specification to define the policies for authorization.

The access control of a resource is defined either by its accompanying ACL file or by the access control inherited by its parent. This means that when parents recursively do not have an ACL file, the Solid Pod root ACL file is followed.

To allow a certain person or service access to read sensor data from a collection, the whole set of resources describing the collection (i.e. the platform metadata, the root node TREE fragment and all the TREE fragments which can be found in the relations of the root node), must be given an ACL file. In each of those ACL files, an *acl:Authorization* schema is added that consists of the read mode and the agent, which is the WebID of the person or service to which access is given to. Listing 4.4 provides this schema, marked with *<#auth>*. Note that the owner of the Pod always explicitly has all access modes.

```
1 @prefix : <#>.
2 @prefix acl: <http://www.w3.org/ns/auth/acl#>.
3 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
```

```
4
5 <#auth> a acl:Authorization;
6     acl:accessTo <./>:;
7     acl:agent <WebID>;
8     acl:default <./>;
9     acl:mode acl:Read.
10
11 <#ControlReadWrite>
12     a acl:Authorization;
13     acl:accessTo <./>;
14     acl:agent <profile/card/#me>;
15     acl:default <./>;
16     acl:mode acl:Control, acl:Read, acl:Write.
```

Listing 4.4: Read acces to a resource

A simple algorithm is implemented that uses four arguments: the root node URL, credentials, the WebID and the access mode. The credentials must be the owner credentials of the Pod or credentials which have *acl:Control*, *acl:Read* and *acl:Write* authorization of the whole collection and platform. Using those arguments, ACL files are added containing valid permissions to each file of the collection and to the platform.

Algorithm 3: Storing a TREE collection of sensor data on a shape tree supported LDP

```
Input: Sensor data in SOSA format
sensorData = sensor data translated to SOSA
LDPURL                                          // URL of the Linked Data Platform
shapeTreeURL                                    // URL where the Shape Tree file is hosted
Function StoreCollectionToLDP(sensorData, LDPURL, shapeTreeURL):
    rootURL = PlantShapeTree(LDPURL, shapeTreeURL, "platform_collection")  // see
     Algorithm 1
    platformName = unique platform name extracted from sensorData
    uniquePlatformInstance = HTTP GET platformURL+platformName
    if uniquePlatformInstance does not exist then
        Create Data Instance using sensorData, rootURL and platformName
    else
        sensorList = []       // List of sensor metadata triples that must be added
        collectionList = []          // List of all triples that form the collection
        for sensor in sensorData do
            if sensor not in uniquePlatformInstance then
                sensorList.add(sensor metadata)
            /* The observation and result are from the current sensor          */
            collectionList.add(Observation as a member)
            collectionList.add(Observation and result)
        if length(sensorList) > 0 then
            Patch platform using sensorList, rootUrl and platformName
        if today collection not exists then
            Post today collection
        else
            Patch today collection
        if root collection not exists then
            create root collection and post
        Patch Data Instance using updateList, rootURL and platformName
```

Algorithm 4: Update the root node

```
Function UpdateRoot(rootURL,timestamp):
    rootGraph = get root file using rootURL
    extract todayRelation, last7daysRelation and currentMonthRelation from rootGraph
    change tree:value to timestamp in todayRelation
    if timestamp day = 1 then
        add archive month to rootGraph // The archive month is based on the month
            before timestamp
        remove last7daysRelation and currentMonthRelation from the rootGraph
    else if timestamp day < 9 then
        change tree:value to (timestamp of first day in month) in last7daysRelation
        remove currentMonthRelation from the rootGraph
    else
        change tree:value to (timestamp - 7 days) in last7daysRelation
        change tree:value to (timestamp of first day in month) in currentMonthRelation
    return rootGraph
```

Algorithm 5: Update the root node

```
Function FragmentCollection(rootURL):
    rootGraph = get root file using rootURL
    extract todayRelation, last7daysRelation, currentMonthRelation and the most recent
     archiveMonthRelation from rootGraph
    move all members older than todayRelation tree:value from todayFragment to
     last7daysFragment
    if last7daysRelation present then
        move all members older than last7daysRelation tree:value from last7daysFragment to
         currentMonthFragment
    else
        move all members from last7daysFragment to currentMonthFragment
    if currentMonthRelation present then
        // Nothing happen here as all members are younger than
            currentMonthRelation tree:value
    else
        if currentMonthFragment not empty then
            move all members from currentMonthFragment to archiveMonthFragment
```

Algorithm 6: Extraction of members which will be removed based on the retention policies

Input: A tree:Collection

Output: A list of members, which according to the policies should be removed

policies = list consisting of parsed policies from the collection

members = list consisting of all members of the collection

```
Function ExtractMembers(policies, members):
    untouchedList = Set(members)
    keepList = []
    for policy in policies do
        categories = memberCategories(policy, members)
        untouchedList = untouchedList ∩ categories.untouched
        keepList = keepList ∪ categories.keep
    return members (keepList ∪ untouchedList)

Function memberCategories(policy, members):
    switch policy do
        untouchedList = []
        keepList = []
        case Time-based Retention Policy do
            duration = parse duration from tree:value
            for member in members do
                memberTime = extract timestamp using tree:path
                if memberTime > currentTime - duration then
                    keepList.add(member)
            break
        case Version-based Retention Policy do
            subset = members part of of the set based on ldes:VersionKey
            untouchedList = members subset
            amount = parse amount from tree:amount
            amountToRemove = length(subset) - amount
            sortedSubset = sort subset from oldest newest based on tree:path
            if amountToRemove > 0 then
                keepList = sortedSubset[amountToRemove:]
            break
    return {keepList, untouchedList}
```

Algorithm 7: Execute the retention policies of a collection

```
Function ExecutePolicy(rootURL):
    memberFileMap= {}
    rootGraph = get root file using rootURL
    members = []
    policies = extract the policies from the root node
    for relation in rootGraph do
        collectionURL = retrieve from the relation
        collection = get collection using collectionURL
        for member in collectionGraph do
            memberFileMap[member] = collectionURL
            members.add(member)
    removeList = ExtractMembers(policies,members)
    removePerFile = {}
    for member in removeList do
        collectionURL = memberFileMap[member]
        add member to list in removePerFile[collectionURL]
    for URL in removePerFile.keys do
        membersToRemove = removePerFile[URL]
        Patch to URL to remove membersToRemove
```

5 Evaluation

In this chapter, the proposed solution is evaluated in terms of scalability. The goal of the evaluation is to verify if fragmentation indeed reduces the time to add new observations to the storage. Furthermore, the impact of preserving validated files on an LDP that keeps growing in volume is quantified.

5.1 Setup

The test setup consists of two servers. One server hosts the LDP with Shape Tree support, the other one hosts the RML Mapper.

The LDP is not a Solid Pod, as Solid currently does not support Shape Trees. However, since authentication does not have an impact on the result, an LDP can be used to evaluate the proposed solution. The RML Mapper only has to generate a session token during authentication once. For every operation on the Pod, the same token can be used. Verifying the token does take time, but for every test performed, this time is the same. Thus, instead of a Solid Pod an adapted version of the shapetrees test-suite [50] is used.

The performance of the solution is tested by comparing three configurations. The first configuration stores the sensor data using the SOSA model to a single file on the LDP. The second

configuration also stores data to a single file, which is now represented by a TREE collection using the SOSA model. The final configuration uses the same strategy as the second one, however, the data is fragmented at a given interval.

5.2 Performance of fragmentation strategy on adding data

In order to compare the three configurations, the RML Mapper must handle the same amount of messages from the broker. Therefore, a first burst mode test was executed by publishing 50 measurements to the broker with a delay of 1 second between consecutive measurements. Figures 5.1, 5.2 and 5.3 show the results of the plain SOSA model, the TREE collection and the TREE collection with fragmentation during this burst mode. The y-axis represents the

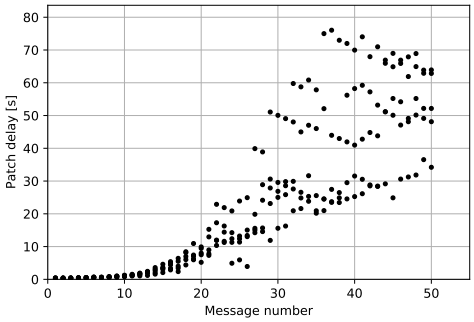

Figure 5.1: SOSA model in burst mode

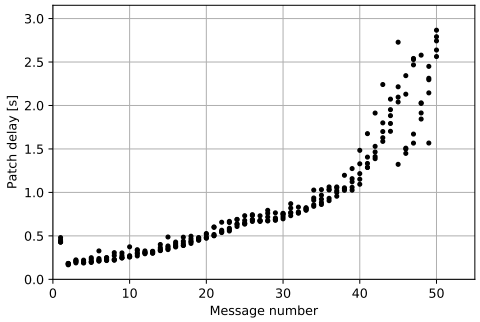

Figure 5.2: TREE collection with SOSA model in burst mode

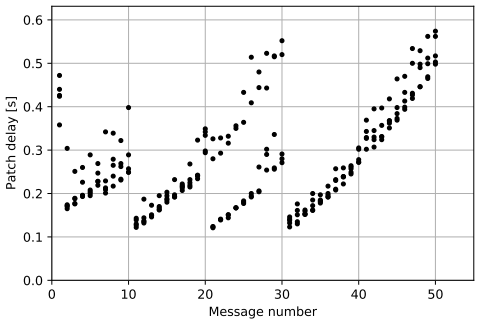

Figure 5.3: TREE collection with SOSA model in burst mode, fragmenting every 10 measurements

patch delay. This is the time it takes to add the measurements of a single message. The x-axis represents the n-th addition to the LDP. On the first two figures, unexpected behavior starts occurring when the patch delay is bigger than 1 second. From then on, some messages are not

stored on the LDP, and also the time per PATCH request starts taking significant more time. However, when fragmentation is executed, this behavior does not occur, as demonstrated in Figure 5.3. The reason why the delay becomes bigger than 1 second is because the validation adds a significant overhead. An analysis on the impact of data validation before storing the data is further elaborated in Section 5.3.

In a second experiment, the same amount of messages are published, however the time between two successive messages is increased. The x-axis now represents the total size of the file after a patch. Figure 5.4 shows that adding new measurements to the plain SOSA model when the stored size surpasses 400 kB takes more than nine seconds. Contrary, a PATCH request to a TREE collection takes only one second as is shown in Figure 5.5.

The difference in performance between adding measurements to plain SOSA or to a TREE collection is mainly caused by a more complex validation strategy for the single SOSA model. Though, the number of measurements keeps growing over time. As the stored size grows, adding new measurements take more time as well. The conclusion is that appending measurements to a single file without a further optimization is not feasible.

In Figure 5.6, a fragmentation strategy is used as an optimization. There, the fragmentation service is executed after every 10 measurements added. As a result, the stored fill size is being limited to 120 kB and a single PATCH only takes 0.35 seconds.

5.2.1 Analysis for the optimal fragmentation strategy

In order to find the optimal fragmentation strategy, a scenario where a sensor measures its environment every ten minutes is used. As a result, every day 144 measurements are added to the LDP.

The following strategies were considered: fragmenting every 10, 20 and 50 measurements and fragmenting once every day. Figures 5.7, 5.8, 5.9 and 5.10 visualize the results.

The first three strategies all store new measurements to the LDP in less than two seconds. For the last strategy all the patches were completed within 15 seconds. This strategy, albeit being slower, is still fast enough for current scenario.

Testing the TREE configurations showed that the patch delay when adding measurements to TREE collection has a quadratic behaviour in function of the file storage size. Furthermore, Figure 5.10 shows that when the file storage size exceeds 800 kB, due to the quadratic behavior, the patch delay increases significantly. All additions before this mark are added within three seconds to the LDP. Consequently, a fragmentation strategy is advised which limits the storage size to 800kB. This means an upper bound for the fragmentation strategy for this setup, where

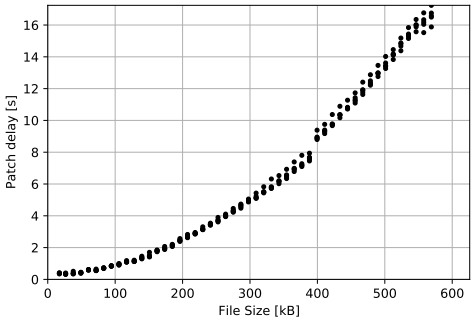

Figure 5.4: SOSA model: stored file size versus delay

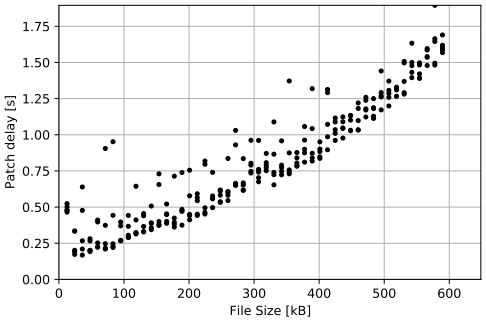

Figure 5.5: TREE collection with SOSA model: stored file size versus delay

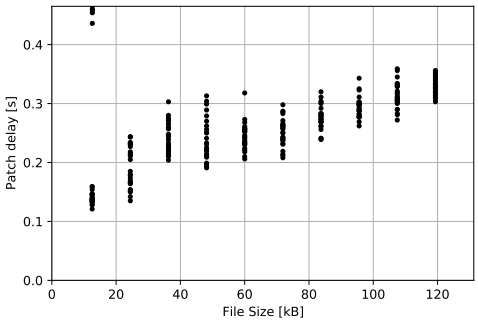

Figure 5.6: TREE collection with SOSA model: stored file size versus delay, fragmenting every 10 measurements

each additional measurement is 12kB, is every 65 measurements.

An attempt to define the lower bound is made by analyzing the fragmentation times of those different strategies, which can be seen in Figure 5.11. The fragmentation strategy measuring every ten added measurements is represented by the blue dots, every 20 measurements by orange dots, every 50 measurements by green dots and finally fragmenting once a day by the red dot. However, for all tests the LDP was initialized, i.e. there was no data on the LDP at the start of the each test. In order to measure the fragmentation impact during run-time, the fragmentation time has to be calculated under the assumption that the last 7 days file contains measurements from the last 7 days. This translates to 1008 measurements being present, for a total of approximately 11900 kB. It should be noted that after the eight of the month, once a day data will be added to the current month. Maximally 24 days of measurements are stored there, which corresponds to 3456 measurements or ∼40750 kB. And once a month, storing corresponds to 4464 measurements. Extrapolating the blue and orange dots of Figure 5.11 results in two quadratic formulas. Using those formulas, the fragmentation delays for the number of measurements are calculated and

shown in table 5.1.

	Fragmenting every 10 measurements	Fragmenting every 20 measurements
1008 measurements	587	641
3456 measurements	6948	7641
4464 measurements	11598	12764

Table 5.1: Extrapolation of the fragmentation delay as a function of the number of measurements stored.

For fragmenting every 10 measurements, there is most of the time a fragmentation delay of 587 seconds, indicating that, because the service is executed every 100 minutes, 10% of time the service is fragmenting. When the same calculation is applied to fragmenting every 20 measurements, the service is fragmenting for about %5. The third and final strategy do not have enough data points to extrapolate a formula.

The optimal fragmentation strategy for current setup is when fragmenting between 20 and 65 measurements.

5.3 Analysis of validation on storing data

In the previous section, a notable difference was observed when adding data to the LDP with the SOSA model versus the TREE collection using SOSA.

In order to test whether validation has an impact when adding new measurements, the same test from the beginning from Section 5.2 was executed. The results are shown Figures 5.12,5.13 and 5.14. The test shows that, when no validation is executed by the LDP, it takes the same amount of time to PATCH data for either configuration. All of them are executed within 0.7 seconds, showing that validation has a big impact on storing to one file.

This raises the question on why validating plain SOSA data takes significantly more time. The answer to this question is that the plain SOSA model has a cyclic reference between a *sosa:Observation* and *sosa:Sensor*, which is not present in the TREE collection. In the TREE collection the observations do reference the sensors, but not vice versa.

Another conclusion is that the patch delay times of the validated and non validated case for the configuration which fragments behave similar. This indicates that fragmenting is required to reduce this validation delay.

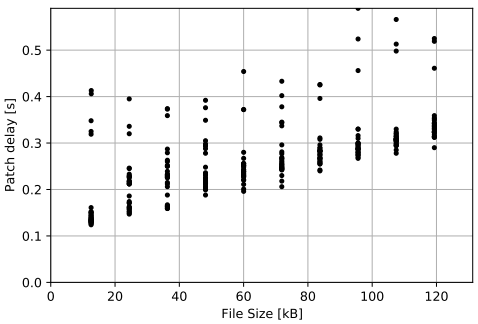

Figure 5.7: Every 10 additions fragmentation strategy: stored file size versus delay

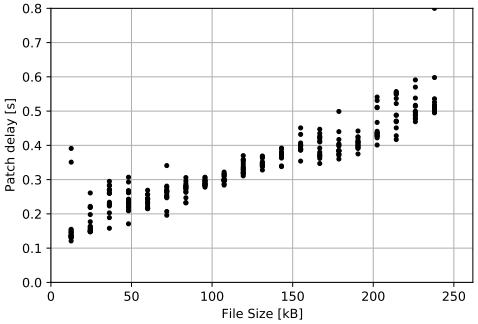

Figure 5.8: Every 20 additions fragmentation strategy: stored file size versus delay

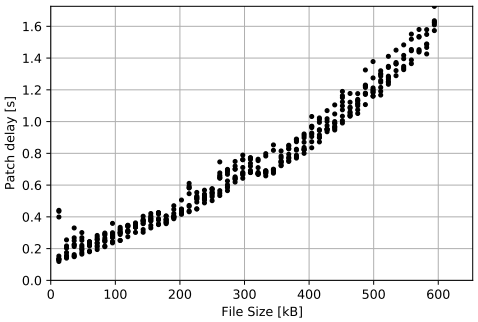

Figure 5.9: Every 50 additions fragmentation strategy: stored file size versus delay

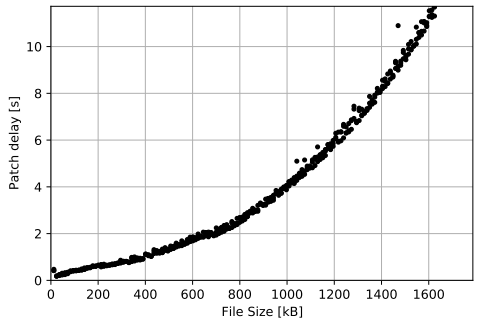

Figure 5.10: Once per day fragmentation strategy: stored file size versus delay

5.4 Impact of the retention policy

Data volume cannot grow infinitely, while sensors measure constantly and some data is not useful. Therefore, a retention service was implemented conforming to the LDES specification.

Using the version-based policy in Listing 5.1, the service was evaluated. Figure 5.15 shows the results when executing the service every 10 measurements and for Figure 5.16 the service was executed every 20 measurements. The figures show that the file size does not grow in volume after a certain threshold, 450kB and 550kB respectively, due to the retention policy.

```
1 <policy> a <https://w3id.org/ldes#LatestVersionSubset>;
2     <https://w3id.org/tree#path> sosa:resultTime;
3     <https://w3id.org/ldes#amount> "500"^^xsd:integer.
```

Listing 5.1: Version based policy that allows for 500 measurements to be stored

Figure 5.15 shows the and 5.16

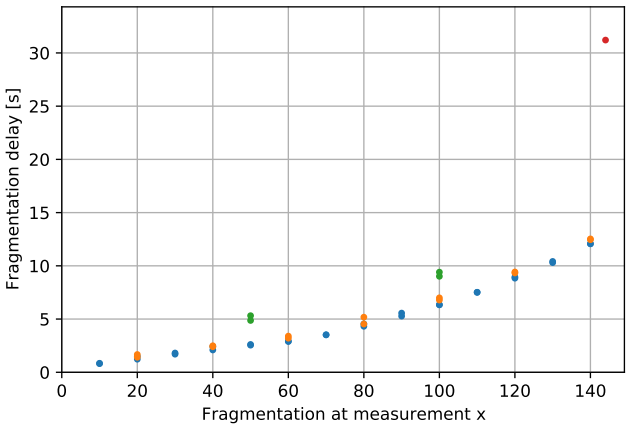

Figure 5.11: Fragmentation times versus total number of measurements in the LDP for the different fragmentation configurations

Defining policies makes managing the data volume on the LDP automatic, while guaranteeing that data that should be preserved, stays on an LDP.

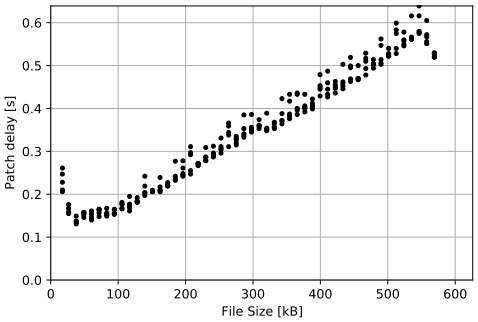

Figure 5.12: SOSA model in burst mode without validation

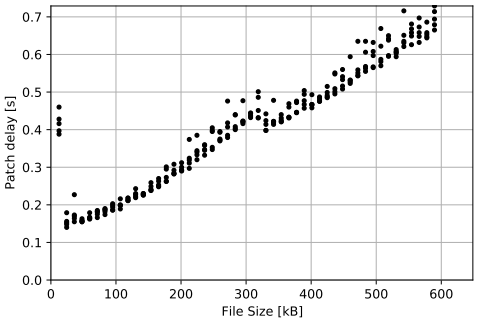

Figure 5.13: Tree collection with SOSA model in burst mode without validation

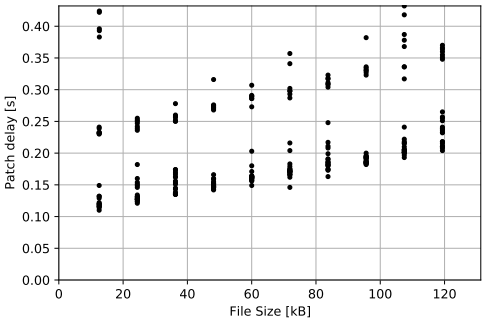

Figure 5.14: Tree collection with SOSA model in burst mode without validation, fragmenting every 10 measurements

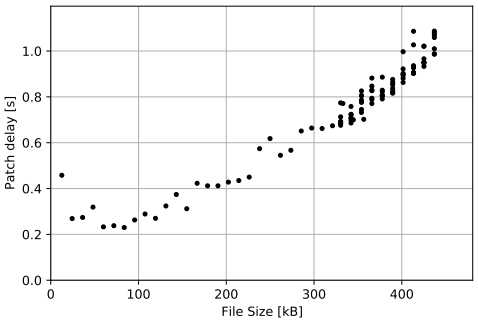

Figure 5.15: File size versus patch delay while running the retention policy every 10 measurements

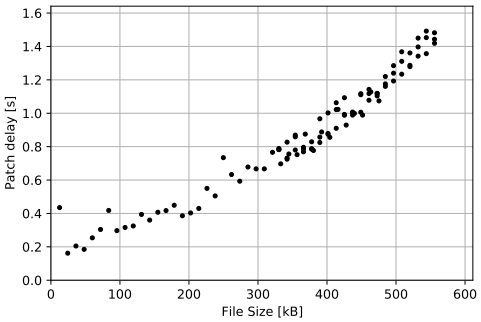

Figure 5.16: File size versus patch delay while running the retention policy every 10 measurements

6

Conclusion

In this thesis, a solution was designed for storing sensor data in an interoperable way to a Linked Data Platform. Storing the data to Solid, a novel type of LDP, allows the end user to have full control over that data. Furthermore, research was done to allow the volume of sensor data to scale in size.

An architecture based on open standards has been designed and implemented involving the processing steps from data generation to storage, together with two services. The first service, the fragmentation service, improves the speed of storing new measurements to the dataset. The retention service manages the total volume of the dataset by executing user defined policies.

The solution demonstrates how to model sensor data and its semantics with linked data and how to validate it using a shape. Using Shape Trees, resources were linked to their corresponding shape. The shape is thus the interface which can be used to build applications on top of the data. The proposed solution avoids the need for defensive programming of applications as only validated data is stored on the platform.

It is shown that the time to validate files is quadratics related to the size. To overcome this problem, two services are implemented and tested. Both succeed in reducing the time to add more data to the pod.

The fragmentation service was evaluated and was showed to be mandatory for handling and

storing a big amount of measurements. An optimal fragmentation strategy is proposed being a trade off between the time for storing new measurements and the time for completing the fragmentation.

Furthermore, the retention policy was evaluated. The impact of this service is that the size of the dataset can be automatically controlled.

7

Future work

Here, recommendations are made to have a starting point for solving the current limitations.

7.1 Shape Tree supported Solid Pod

The validations takes place on a local LDP and are later copied to a Solid Pod. This is not efficient as data is sent to two LDPs. Storing new measurements directly to the Pod would improve the delay. When Solid Pods support Shape Trees, some adaptations can therefore be made to remove the step where data is stored to the LDP.

7.2 Burst data

A sensor with a higher rate for sending measurements, poses a problem for storing the data. In such situations, the validation takes too long. Two paths are proposed which might, especially when combined, pose as a solution to this problem.

The first solution is a batch approach. The idea is to aggregate the data in a buffer in the RML Mapper for a certain time window. At the end of each time window, the new measurements are

sent to the LDP which then validates the data.

A second approach is to improve the fragmentation strategy. The current smallest TREE fragment file collects all measurements for a given day. When defining the TREE fragment to contain one hour of measurements, the total size per file is reduced. Another improvement is to change the way archiving works. Currently, this was done at the beginning of the month by storing all the data of the previous month to one file. The evaluation showed that this file is big. Too reduce that size, the functionality of the today file remains. But fragmenting would result in directly archiving it to a file which again contains one hour of measurements.

7.3 Improve the retention service

Executing the retention policy results in removing some the oldest members. However, the service does not know how many members are located in each file. This means that for executing a version subset, every file has to be retrieved. A solution to this problem is to add to each relation of the root node a triple with the *tree:remainingItems* predicate. The TREE Hypermedia specification [10] designed this predicate to define how many members of the collection of the node of the relation.

Another addition is to add the total amount of members in the collection to the root node. This can be done by adding a triple with predicate *hydra:totalItems* defined by the Hydra Collections specifications [69] as this specification is compatible with the TREE specification.

7.4 Enforce members in an event stream to be immutable

The core principle of LDES is that its members are immutable. With Solid however, the owner has write and read permission over the files. Which means he/she could alter some members. It would be useful to research the capabilities of a Solid Pod and the WAC/ACL specification to enforce immutability of LDES members. This should also take into account that no members of the collection can be removed if the retention policies do not allow it.

Bibliography

[1] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic Sensor Network Ontology," Oct. 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/

[2] S. Speicher, J. Arwe, and A. Malhotra, "Linked Data Platform 1.0," Feb. 2015. [Online]. Available: https://www.w3.org/TR/ldp/

[3] "LoRa Architecture - LoRaWAN Tutorial." [Online]. Available: https://www.3glteinfo.com/lora/lora-architecture/

[4] F. Cirillo, F.-J. Wu, G. Solmaz, and E. Kovacs, "Embracing the Future Internet of Things," *Sensors*, vol. 19, no. 2, p. 351, Jan. 2019. [Online]. Available: http://www.mdpi.com/1424-8220/19/2/351

[5] J.-L. Gassée, "Internet of Things: The "Basket of Remotes" Problem," Mar. 2016. [Online]. Available: https://mondaynote.com/internet-of-things-the-basket-of-remotes-problem-f80922a91a0f

[6] Z. Aufort, "Sharing Credentials is a Really Bad Idea. Seriously, Don't Do It." Oct. 2020. [Online]. Available: https://puget.tech/sharing-credentials-bad-idea/

[7] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, J. Bingham, and D. Zagidulin, "The Solid Protocol." [Online]. Available: https://solidproject.org/TR/protocol#authentication

[8] A. Sambra, H. Story, and T. Berners-Lee, "WebID 1.0," Mar. 2014. [Online]. Available: https://www.w3.org/2005/Incubator/webid/spec/identity/

[9] E. Prud'hommeaux and J. Bingham, "Shape Trees Specification." [Online]. Available: https://shapetrees.org/TR/specification/

[10] P. Colpaert, "The TREE hypermedia specification," Feb. 2021. [Online]. Available: https://treecg.github.io/specification/

[11] ——, "Linked Data Event Streams spec," Feb. 2021. [Online]. Available: https://semiceu.github.io/LinkedDataEventStreams/eventstreams.html

[12] "Semantics," May 2021, page Version ID: 1021847353. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Semantics&oldid=1021847353

[13] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001, publisher: JSTOR.

[14] T. Berners-Lee, "Linked Data - Design Issues," Jul. 2006. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[15] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC Editor, Tech. Rep. RFC3986, Jan. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc3986

[16] P. Saint-Andre and J. Klensin, "Uniform Resource Names (URNs)," RFC Editor, Tech. Rep. RFC8141, Apr. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8141

[17] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC Editor, Tech. Rep. RFC1738, Dec. 1994. [Online]. Available: https://www.rfc-editor.org/info/rfc1738

[18] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carrol, and B. McBride, "RDF 1.1 Concepts and Abstract Syntax," 2014. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[19] E. Prud'hommeaux, G. Carothers, T. Berners-Lee, and D. Becket, "RDF 1.1 Turtle." [Online]. Available: https://www.w3.org/TR/2014/REC-turtle-20140225/

[20] G. Kellog, M. Sporny, M. Lanthaler, D. Longley, and N. Lindström, "JSON-LD 1.1." [Online]. Available: https://json-ld.org/spec/latest/json-ld/

[21] A. Dhörthe, W. Van Woensel, and D. Tomaszuk, "Notation3." [Online]. Available: https://w3c.github.io/N3/spec/

[22] F. Gandon, G. Schreiber, and D. Becket, "RDF 1.1 XML Syntax." [Online]. Available: https://www.w3.org/TR/rdf-syntax-grammar/

[23] A. Haller, K. Janowicz, S. J. Cox, M. Lefrançois, K. Taylor, D. Le Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, and C. Stadler, "The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation," *Semantic Web*, vol. 10, no. 1, pp. 9–32, Dec. 2018. [Online]. Available: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-180320

[24] "World Wide Web Consortium (W3C)." [Online]. Available: https://www.w3.org/

[25] "The Home of Location Technology Innovation and Collaboration | OGC." [Online]. Available: https://www.ogc.org/

[26] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "SOSA: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, vol. 56, pp. 1–10, May 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1570826818300295

[27] "Foodvoc." [Online]. Available: http://www.ontology-of-units-of-measure.org/

[28] F. Team, "FAIRsharing record for: Quantities, Units, Dimensions and Types," 2015, type: dataset. [Online]. Available: https://fairsharing.org/FAIRsharing.d3pqw7

[29] "Generiek Basis." [Online]. Available: https://data.vlaanderen.be/doc/applicatieprofiel/generiek-basis/#Standaard%20Eenheid

[30] R. Buyle, L. De Vocht, M. Van Compernolle, D. De Paepe, R. Verborgh, Z. Vanlishout, B. De Vidts, P. Mechant, and E. Mannens, "OSLO: open standards for linked organizations," in *Proceedings of the International Conference on Electronic Governance and Open Society: Challenges in Eurasia*. St. Petersburg Russia: ACM, Nov. 2016, pp. 126–134. [Online]. Available: https://dl.acm.org/doi/10.1145/3014087.3014096

[31] L. Daniele, F. den Hartog, and J. Roes, "Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology," in *Formal Ontologies Meet Industry*, R. Cuel and R. Young, Eds. Cham: Springer International Publishing, 2015, vol. 225, pp. 100–112, series Title: Lecture Notes in Business Information Processing. [Online]. Available: http://link.springer.com/10.1007/978-3-319-21545-7_9

[32] L. Daniele, R. Garcia-Castro, M. Lefrançois, and M. Poveda-Villalon, "SAREF: the Smart Applications REFerence ontology." [Online]. Available: https://saref.etsi.org/core/v3.1.1/

[33] J. L. Moreira, L. M. Daniele, L. Ferreira Pires, M. J. van Sinderen, K. Wasielewska, P. Szmeja, W. Pawlowski, M. Ganzha, and M. Paprzycki, "Towards IoT platforms' integration: Semantic Translations between W3C SSN and ETSI SAREF," Nov. 2017.

[34] V. Charpenay, S. Kabisch, and H. Kosch, "Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things," p. 12.

[35] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay, and M. Kovatsch, "Web of Things (WoT) Thing Description," Apr. 2020. [Online]. Available: https://www.w3.org/TR/wot-thing-description/

[36] E. Prud'hommeaux, "SPARQL Query Language for RDF," Jan. 2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[37] E. Prud'hommeaux, A. Seaborne, and S. Harris, "SPARQL 1.1 Query Language," Mar. 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[38] R. Taelman, J. Van Herwegen, M. Vander Sande, and R. Verborgh, "Comunica: a Modular SPARQL Query Engine for the Web," in *Proceedings of the 17th International Semantic Web Conference*, ser. Lecture Notes in Computer Science, D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, Eds., vol. 11137. Springer, Oct. 2018, pp. 239–255. [Online]. Available: https://comunica.github.io/Article-ISWC2018-Resource/

[39] R. Verborgh, "Linked Data Fragments." [Online]. Available: https://linkeddatafragments.org/specification/linked-data-fragments/

[40] N. Mihindukulasooriya and R. Menday, "Linked Data Platform 1.0 Primer," Apr. 2015. [Online]. Available: https://www.w3.org/TR/2015/NOTE-ldp-primer-20150423/

[41] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape expressions: an RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems - SEM '14*. Leipzig, Germany: ACM Press, 2014, pp. 32–40. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2660517.2660523

[42] E. Prud'hommeaux, I. Boneva, J. E. L. Gayo, and G. Kellog, "Shape Expressions Language 2.1." [Online]. Available: http://shex.io/shex-semantics/

[43] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," Jul. 2017. [Online]. Available: https://www.w3.org/TR/shacl/

[44] A. Le Hors, E. Prud'hommeaux, S. Hawke, I. Polikoff, and T. Thibodeau Jr, "W3C RDF Data Shapes Working Group Charter," 2014. [Online]. Available: https://www.w3.org/2014/data-shapes/charter

[45] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 7, no. 1, pp. 1–328, Sep. 2017. [Online]. Available: http://www.morganclaypool.com/doi/10.2200/S00786ED1V01Y201707WBE016

[46] T. Baker and E. Prud'hommeaux, "Shape Expressions (ShEx) 2.1 Primer," Oct. 2019. [Online]. Available: http://shex.io/shex-primer/

[47] "ShEx2 — Simple Online Validator." [Online]. Available: https://rawgit.com/shexSpec/shex.js/master/packages/shex-webapp/doc/shex-simple.html

[48] J. Bingham, E. Prud'hommeaux, and J. Collins, "Solid Application Interoperability." [Online]. Available: https://solid.github.io/data-interoperability-panel/specification/

[49] "shapetrees/shapetree.js-old," Feb. 2021, original-date: 2020-04-02T09:00:22Z. [Online]. Available: https://github.com/shapetrees/shapetree.js-old

[50] "shapetrees/test-suite," Apr. 2021, original-date: 2020-01-10T13:51:24Z. [Online]. Available: https://github.com/shapetrees/test-suite

[51] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: A Platform for Decentralized Social Applications Based on Linked Data," p. 16, 2016.

[52] P. d. Poulpiquet, "What is a Walled Garden? And why it is the strategy of Google, Facebook and Amazon Ads platform?" Nov. 2017. [Online]. Available: https://medium.com/mediarithmics-what-is/what-is-a-walled-garden-and-why-it-is-the-strategy-of-google-facebook-and-amazon-ads-platform-296ddeb784b1

[53] A. Coburn, E. Pavlik, and D. Zagidulin, "SOLID-OIDC." [Online]. Available: https://solid.github.io/authentication-panel/solid-oidc/

[54] "Web Access Control (WAC)." [Online]. Available: http://solid.github.io/web-access-control-spec/

[55] "Access Control ontology." [Online]. Available: https://www.w3.org/ns/auth/acl#

[56] F. Sanders, "Solid Pods for IoT," 2021.

[57] "TREEcg/treemunica_typeahead_demo," Apr. 2021, original-date: 2021-02-17T17:08:18Z. [Online]. Available: https://github.com/TREEcg/treemunica_typeahead_demo

[58] D. G. Edkins, "Human Factors, Human Error & The Role of Bad Luck in Incident Investigations," May 2016. [Online]. Available: https://www.safetywise.com/single-post/2016/08/30/human-factors-human-error-the-role-of-bad-luck-in-incident-investigations

[59] A. Dimou, M. Vander Sande, B. De Meester, P. Heyvaert, and T. Delva, "RDF Mapping Language (RML)." [Online]. Available: https://rml.io/specs/rml/

[60] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF Mapping Language." [Online]. Available: https://www.w3.org/TR/r2rml/

[61] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low Power Wide Area Networks: A Survey," *CoRR*, vol. abs/1606.07360, 2016, _eprint: 1606.07360. [Online]. Available: http://arxiv.org/abs/1606.07360

[62] N. Sornin, M. Luis, T. Eirich, and T. Kramp, "LoRaWAN® Specification v1.1." [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/

[63] "MQTT - The Standard for IoT Messaging." [Online]. Available: https://mqtt.org/

[64] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE*

Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7123563/

[65] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," *The Journal of Open Source Software*, vol. 2, no. 13, p. 265, May 2017. [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00265

[66] "eclipse/mosquitto," May 2021, original-date: 2016-03-10T20:19:09Z. [Online]. Available: https://github.com/eclipse/mosquitto

[67] "ChirpStack open-source LoRaWAN® Network Server." [Online]. Available: https://www.chirpstack.io/

[68] "solid/node-solid-server," May 2021, original-date: 2014-09-21T22:05:03Z. [Online]. Available: https://github.com/solid/node-solid-server

[69] M. Lanthaler, "Hydra Core Vocabulary." [Online]. Available: https://www.hydra-cg.com/spec/latest/core/#introduction

Appendices

Appendix A - HTTP URL

The syntax of an HTTP URL is the the following:

http://<host>/<path>?<query>#<fragment>

The URL scheme is the HTTP protocol. This is followed by a semicolon and two slashes.

The host starts with either an IP address, which can be both IPv4 and IPv6 (the IPv6 address must be placed inside brackets) or a domain name. That domain name will be resolved by a Domain Name System (DNS). Optionally a port can be provided to address by adding a semicolon and the port number.

The path follows the host with a slash. It indicates the path to the resource, which is hosted on the server.

In the query part it is possible to define extra parameters. They are key-value pairs seperated by an ampersant (&) symbol.

Finally there is the fragment, sometimes named anchor. The fragment begins with a hash (#) and is followed by an identifier and is used to refer to a part of the resource. With a fragment a browser can show that part of web page directly. The fragment is not send in the request to the server.

The secure version of HTTP, HTTPS, its syntax very similar. The only difference is that the ”http” scheme is replaced by ”https”.

https://<host>/<path>?<query>#<fragment>

Appendix B - Sensor data received from MQTT

The data below 1, represented with JSON, is an example of a packet received by a service that is subscribed to the specific topic of the MQTT.

The actual measurements of the data can be retrieved from the ”data” value of the object ”device-ctrl”. This value corresponds with:
”AhevAAOAE4AAgACAAIBpiqaBNICBgFam+YI/gIOAGH/hgACABX+XCmU=”.

B.1 MQTT sensor data packet


```
{
    "device-ctrl": {
        "dev-eui": "0004a30b0024e96c",
        "data": "AhevAAOAE4AAgACAAIBpiqaBNICBgFam+YI/gIOAGH/hgACABX+XCmU=",
        "device-custom-ctrl": {
            "applicationID": "1",
            "applicationName": "port-fw",
            "deviceName": "decentlab-weather-station",
            "rxInfo": [
                {
                    "gatewayID": "1dee0be9d4472aaf",
                    "uplinkID": "270bb8f5-ccb0-4382-abb6-11e2a9774690",
                    "name": "LoRank8",
                    "rssi": -67,
                    "loRaSNR": 10,
                    "location": {
                        "latitude": 0,
                        "longitude": 0,
                        "altitude": 0
                    }
                }
            ],
            "txInfo": {
                "frequency": 868300000,
                "dr": 5
            },
            "adr": false,
            "fCnt": 26,
```

```
            "fPort": 1
        }
    },
    "network-ctrl": {
        "network-type": "loRaWAN",
        "network-custom-ctrl": {
            "src-protocol-ctrl": {
                "sub-topic": "application/+/device/+/event/up",
                "pub-topic": "application/+/device/+/command/down"
            },
            "fPort": 1,
            "dr": 5,
            "frequency": 868300000,
            "rssi": -67,
            "snr": 10
        }
    },
    "output-ctrl": {
        "device-id": 3,
        "ipv6": "2001:6a8:1d80:2031:225:90ff:fe4c:77e0"
    },
    "central-broker-ctrl": {
        "central-broker-topic-ctrl": {
            "pub-topic": "proc",
            "pub-topic-direction": "ul",
            "sub-topic": "data",
            "sub-topic-direction": "ul"
        },
        "adapter-ctrl": {
            "adapter-id": 3
        }
    }
}
```

Appendix C - Intermediate JSON representation of sensor data

Below (1 is the intermediate representation of the sensor data from the packet in Appendix B.

The JSON object contains the UUID value of the sensor, the given name of the sensor and a list of measurements. Apart from the decoded value, each measurement is accompanied by metadata. This metadata consists of the name of what is being measured, unit definitions from the QUDT ontology, a dateTime object representing the time when it was received in the RML Mapper and the *sosa:observableProperty*, which defines with the SOSA model what is being observed.

C.1 IR of the sendor data


```
{
    "uuid": "0004a30b0024e96c",
    "name": "decentlab-weather-station",
    "measurements": [
        {
            "property": "solar_radiation",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": 19,
                "unit": "W-PER-M2",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "solarRadiation"
            }
        },
        {
            "property": "precipitation",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": 0,
                "unit": "MilliM",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "precipitation"
            }
        },
        {
```

```
        "property": "lightning_strike_count",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 0,
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "lightningStrikeCount"
        }
    },
    {
        "property": "lightning_average_distance",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 0,
            "unit": "KiloM",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "lightningAverageDistance"
        }
    },
    {
        "property": "wind_speed",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 1.05,
            "unit": "M-PER-SEC",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "windSpeed"
        }
    },
    {
        "property": "wind_direction",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 272.6,
            "unit": "DEG",
            "time": "2021-05-28T16:31:08.904Z",
```

```
            "observableProperty": "windDirection"
        }
    },
    {
        "property": "maximum_wind_speed",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 3.08,
            "unit": "M-PER-SEC",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "maximumWindSpeed"
        }
    },
    {
        "property": "air_temperature",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 12.9,
            "unit": "DEG_C",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "airTemperature"
        }
    },
    {
        "property": "vapor_pressure",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 0.86,
            "unit": "KiloPA",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "vaporPressure"
        }
    },
    {
        "property": "atmospheric_pressure",
        "uuid": "0004a30b0024e96c",
```

```
        "observation": {
            "id": 2294,
            "value": 99.77,
            "unit": "KiloPA",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "atmosphericPressure"
        }
    },
    {
        "property": "relative_humidity",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 57.5,
            "unit": "PERCENT_RH",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "relativeHumidity"
        }
    },
    {
        "property": "sensor_temperature_internal",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 13.1,
            "unit": "DEG_C",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "sensorTemperatureInternal"
        }
    },
    {
        "property": "x_orientation_angle",
        "uuid": "0004a30b0024e96c",
        "observation": {
            "id": 2294,
            "value": 2.4,
            "unit": "DEG",
            "time": "2021-05-28T16:31:08.904Z",
            "observableProperty": "xOrientationAngle"
```

```
            }
        },
        {
            "property": "y_orientation_angle",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": -3.1,
                "unit": "DEG",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "yOrientationAngle"
            }
        },
        {
            "property": "compass_heading",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": 0,
                "unit": "DEG",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "compassHeading"
            }
        },
        {
            "property": "north_wind_speed",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": 0.05,
                "unit": "M-PER-SEC",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "northWindSpeed"
            }
        },
        {
            "property": "east_wind_speed",
            "uuid": "0004a30b0024e96c",
            "observation": {
```

```
                "id": 2294,
                "value": -1.05,
                "unit": "M-PER-SEC",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "eastWindSpeed"
            }
        },
        {
            "property": "battery_voltage",
            "uuid": "0004a30b0024e96c",
            "observation": {
                "id": 2294,
                "value": 2.661,
                "unit": "EV",
                "time": "2021-05-28T16:31:08.904Z",
                "observableProperty": "batteryVoltage"
            }
        }
    ]
}
```

Appendix D - Sensor data represented by the SOSA model

This is the sensor data from Appendix B, but now transformed to the SOSA model, represented in Turtle.

```
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix obs: <http://localhost:8080/shapes/ObservableProperties.ttl>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix tree: <http://www.w3.org/ns/shapetree#>.

<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_air_temperature> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_air_temperature_observation2294>;
    sosa:observes obs:airTemperature.
<platform_0004a30b0024e96c.ttl> a sosa:Platform;
    sosa:hosts <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_air_temperature>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_atmospheric_pressure>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_battery_voltage>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_compass_heading>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_east_wind_speed>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_average_distance>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_strike_count>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_maximum_wind_speed>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_north_wind_speed>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_precipitation>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_relative_humidity>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_sensor_temperature_internal>,
        <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_solar_radiation>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_vapor_pressure>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_direction>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_x_orientation_angle>, <
        platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_y_orientation_angle>.
<sensor_0004a30b0024e96c_air_temperature_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_air_temperature_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_air_temperature
        >;
    sosa:observedProperty obs:airTemperature;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
```

```
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_atmospheric_pressure> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_atmospheric_pressure_observation2294>;
    sosa:observes obs:atmosphericPressure.
<sensor_0004a30b0024e96c_atmospheric_pressure_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_atmospheric_pressure_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_atmospheric_pressure>;
    sosa:observedProperty obs:atmosphericPressure;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_battery_voltage> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_battery_voltage_observation2294>;
    sosa:observes obs:batteryVoltage.
<sensor_0004a30b0024e96c_battery_voltage_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_battery_voltage_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_battery_voltage
        >;
    sosa:observedProperty obs:batteryVoltage;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_compass_heading> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_compass_heading_observation2294>;
    sosa:observes obs:compassHeading.
<sensor_0004a30b0024e96c_compass_heading_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_compass_heading_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_compass_heading
        >;
    sosa:observedProperty obs:compassHeading;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_east_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_east_wind_speed_observation2294>;
    sosa:observes obs:eastWindSpeed.
<sensor_0004a30b0024e96c_east_wind_speed_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_east_wind_speed_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_east_wind_speed
        >;
    sosa:observedProperty obs:eastWindSpeed;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_average_distance> a sosa:
    Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_lightning_average_distance_observation2294
```

```
        >;
    sosa:observes obs:lightningAverageDistance.
<sensor_0004a30b0024e96c_lightning_average_distance_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_lightning_average_distance_observation_result2294
        >;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_lightning_average_distance>;
    sosa:observedProperty obs:lightningAverageDistance;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_strike_count> a sosa:Sensor
    ;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_lightning_strike_count_observation2294>;
    sosa:observes obs:lightningStrikeCount.
<sensor_0004a30b0024e96c_lightning_strike_count_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_lightning_strike_count_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_lightning_strike_count>;
    sosa:observedProperty obs:lightningStrikeCount;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_maximum_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_maximum_wind_speed_observation2294>;
    sosa:observes obs:maximumWindSpeed.
<sensor_0004a30b0024e96c_maximum_wind_speed_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_maximum_wind_speed_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_maximum_wind_speed>;
    sosa:observedProperty obs:maximumWindSpeed;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_north_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_north_wind_speed_observation2294>;
    sosa:observes obs:northWindSpeed.
<sensor_0004a30b0024e96c_north_wind_speed_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_north_wind_speed_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_north_wind_speed>;
    sosa:observedProperty obs:northWindSpeed;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_precipitation> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_precipitation_observation2294>;
    sosa:observes obs:precipitation.
```

```
<sensor_0004a30b0024e96c_precipitation_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_precipitation_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_precipitation>;
    sosa:observedProperty obs:precipitation;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_relative_humidity> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_relative_humidity_observation2294>;
    sosa:observes obs:relativeHumidity.
<sensor_0004a30b0024e96c_relative_humidity_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_relative_humidity_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_relative_humidity>;
    sosa:observedProperty obs:relativeHumidity;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_sensor_temperature_internal> a sosa:
    Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294
        >;
    sosa:observes obs:sensorTemperatureInternal.
<sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294> a sosa:Observation;
    sosa:hasResult <
        sensor_0004a30b0024e96c_sensor_temperature_internal_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_sensor_temperature_internal>;
    sosa:observedProperty obs:sensorTemperatureInternal;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_solar_radiation> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_solar_radiation_observation2294>;
    sosa:observes obs:solarRadiation.
<sensor_0004a30b0024e96c_solar_radiation_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_solar_radiation_observation_result2294>;
    sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_solar_radiation
        >;
    sosa:observedProperty obs:solarRadiation;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_vapor_pressure> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:madeObservation <sensor_0004a30b0024e96c_vapor_pressure_observation2294>;
    sosa:observes obs:vaporPressure.
<sensor_0004a30b0024e96c_vapor_pressure_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_vapor_pressure_observation_result2294>;
```

```
136     sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_vapor_pressure>;
137     sosa:observedProperty obs:vaporPressure;
138     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
139 <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_direction> a sosa:Sensor;
140     sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
141     sosa:madeObservation <sensor_0004a30b0024e96c_wind_direction_observation2294>;
142     sosa:observes obs:windDirection.
143 <sensor_0004a30b0024e96c_wind_direction_observation2294> a sosa:Observation;
144     sosa:hasResult <sensor_0004a30b0024e96c_wind_direction_observation_result2294>;
145     sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_direction>;
146     sosa:observedProperty obs:windDirection;
147     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
148 <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed> a sosa:Sensor;
149     sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
150     sosa:madeObservation <sensor_0004a30b0024e96c_wind_speed_observation2294>;
151     sosa:observes obs:windSpeed.
152 <sensor_0004a30b0024e96c_wind_speed_observation2294> a sosa:Observation;
153     sosa:hasResult <sensor_0004a30b0024e96c_wind_speed_observation_result2294>;
154     sosa:madeBySensor <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed>;
155     sosa:observedProperty obs:windSpeed;
156     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
157 <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_x_orientation_angle> a sosa:Sensor;
158     sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
159     sosa:madeObservation <sensor_0004a30b0024e96c_x_orientation_angle_observation2294>;
160     sosa:observes obs:xOrientationAngle.
161 <sensor_0004a30b0024e96c_x_orientation_angle_observation2294> a sosa:Observation;
162     sosa:hasResult <sensor_0004a30b0024e96c_x_orientation_angle_observation_result2294>;
163     sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_x_orientation_angle>;
164     sosa:observedProperty obs:xOrientationAngle;
165     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
166 <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_y_orientation_angle> a sosa:Sensor;
167     sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
168     sosa:madeObservation <sensor_0004a30b0024e96c_y_orientation_angle_observation2294>;
169     sosa:observes obs:yOrientationAngle.
170 <sensor_0004a30b0024e96c_y_orientation_angle_observation2294> a sosa:Observation;
171     sosa:hasResult <sensor_0004a30b0024e96c_y_orientation_angle_observation_result2294>;
172     sosa:madeBySensor <platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_y_orientation_angle>;
173     sosa:observedProperty obs:yOrientationAngle;
174     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
175 <sensor_0004a30b0024e96c_air_temperature_observation_result2294> a qudt:QuantityValue;
176     qudt:Unit unit:DEG_C;
177     qudt:numericValue "12.9"^^xsd:float;
```

```
178     sosa:isResultOf <sensor_0004a30b0024e96c_air_temperature_observation2294>.
179 <sensor_0004a30b0024e96c_atmospheric_pressure_observation_result2294> a qudt:QuantityValue;
180     qudt:Unit unit:KiloPA;
181     qudt:numericValue "99.77"^^xsd:float;
182     sosa:isResultOf <sensor_0004a30b0024e96c_atmospheric_pressure_observation2294>.
183 <sensor_0004a30b0024e96c_battery_voltage_observation_result2294> a qudt:QuantityValue;
184     qudt:Unit unit:EV;
185     qudt:numericValue "2.661"^^xsd:float;
186     sosa:isResultOf <sensor_0004a30b0024e96c_battery_voltage_observation2294>.
187 <sensor_0004a30b0024e96c_compass_heading_observation_result2294> a qudt:QuantityValue;
188     qudt:Unit unit:DEG;
189     qudt:numericValue "0"^^xsd:float;
190     sosa:isResultOf <sensor_0004a30b0024e96c_compass_heading_observation2294>.
191 <sensor_0004a30b0024e96c_east_wind_speed_observation_result2294> a qudt:QuantityValue;
192     qudt:Unit unit:M-PER-SEC;
193     qudt:numericValue "-1.05"^^xsd:float;
194     sosa:isResultOf <sensor_0004a30b0024e96c_east_wind_speed_observation2294>.
195 <sensor_0004a30b0024e96c_lightning_average_distance_observation_result2294> a qudt:
        QuantityValue;
196     qudt:Unit unit:KiloM;
197     qudt:numericValue "0"^^xsd:float;
198     sosa:isResultOf <sensor_0004a30b0024e96c_lightning_average_distance_observation2294>.
199 <sensor_0004a30b0024e96c_lightning_strike_count_observation_result2294> a qudt:QuantityValue
        ;
200     qudt:numericValue "0"^^xsd:float;
201     sosa:isResultOf <sensor_0004a30b0024e96c_lightning_strike_count_observation2294>.
202 <sensor_0004a30b0024e96c_maximum_wind_speed_observation_result2294> a qudt:QuantityValue;
203     qudt:Unit unit:M-PER-SEC;
204     qudt:numericValue "3.08"^^xsd:float;
205     sosa:isResultOf <sensor_0004a30b0024e96c_maximum_wind_speed_observation2294>.
206 <sensor_0004a30b0024e96c_north_wind_speed_observation_result2294> a qudt:QuantityValue;
207     qudt:Unit unit:M-PER-SEC;
208     qudt:numericValue "0.05"^^xsd:float;
209     sosa:isResultOf <sensor_0004a30b0024e96c_north_wind_speed_observation2294>.
210 <sensor_0004a30b0024e96c_precipitation_observation_result2294> a qudt:QuantityValue;
211     qudt:Unit unit:MilliM;
212     qudt:numericValue "0"^^xsd:float;
213     sosa:isResultOf <sensor_0004a30b0024e96c_precipitation_observation2294>.
214 <sensor_0004a30b0024e96c_relative_humidity_observation_result2294> a qudt:QuantityValue;
215     qudt:Unit unit:PERCENT_RH;
216     qudt:numericValue "57.5"^^xsd:float;
217     sosa:isResultOf <sensor_0004a30b0024e96c_relative_humidity_observation2294>.
218 <sensor_0004a30b0024e96c_sensor_temperature_internal_observation_result2294> a qudt:
        QuantityValue;
```

```
219     qudt:Unit unit:DEG_C;
220     qudt:numericValue "13.1"^^xsd:float;
221     sosa:isResultOf <sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294>.
222 <sensor_0004a30b0024e96c_solar_radiation_observation_result2294> a qudt:QuantityValue;
223     qudt:Unit unit:W-PER-M2;
224     qudt:numericValue "19"^^xsd:float;
225     sosa:isResultOf <sensor_0004a30b0024e96c_solar_radiation_observation2294>.
226 <sensor_0004a30b0024e96c_vapor_pressure_observation_result2294> a qudt:QuantityValue;
227     qudt:Unit unit:KiloPA;
228     qudt:numericValue "0.86"^^xsd:float;
229     sosa:isResultOf <sensor_0004a30b0024e96c_vapor_pressure_observation2294>.
230 <sensor_0004a30b0024e96c_wind_direction_observation_result2294> a qudt:QuantityValue;
231     qudt:Unit unit:DEG;
232     qudt:numericValue "272.6"^^xsd:float;
233     sosa:isResultOf <sensor_0004a30b0024e96c_wind_direction_observation2294>.
234 <sensor_0004a30b0024e96c_wind_speed_observation_result2294> a qudt:QuantityValue;
235     qudt:Unit unit:M-PER-SEC;
236     qudt:numericValue "1.05"^^xsd:float;
237     sosa:isResultOf <sensor_0004a30b0024e96c_wind_speed_observation2294>.
238 <sensor_0004a30b0024e96c_x_orientation_angle_observation_result2294> a qudt:QuantityValue;
239     qudt:Unit unit:DEG;
240     qudt:numericValue "2.4"^^xsd:float;
241     sosa:isResultOf <sensor_0004a30b0024e96c_x_orientation_angle_observation2294>.
242 <sensor_0004a30b0024e96c_y_orientation_angle_observation_result2294> a qudt:QuantityValue;
243     qudt:Unit unit:DEG;
244     qudt:numericValue "-3.1"^^xsd:float;
245     sosa:isResultOf <sensor_0004a30b0024e96c_y_orientation_angle_observation2294>.
```

Appendix E - Sensor data represented by a TREE collection with SOSA as model

Below are three files which form the base of the fragmentation. Those files represent the sensor data, as received from the packet in Appendix B, transformed to a TREE collection with SOSA as model.

File 1 Turtle file contains the platforms and the metadata of the sensors. File 2 is the root file, which contains the root collection and node. The root node has a relation to the third file. Finally in file 3, all the members of the collection can be found.

E.1 File containing the metadata of the sensor using SOSA

```
1 @prefix ldp: <http://www.w3.org/ns/ldp#>.
2 @prefix sosa: <http://www.w3.org/ns/sosa/>.
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
4 @prefix qudt: <http://qudt.org/schema/qudt/>.
5 @prefix unit: <http://qudt.org/vocab/unit/>.
6 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
7 @prefix obs: <http://localhost:8080/shapes/ObservableProperties.ttl>.
8 @prefix dcterms: <http://purl.org/dc/terms/>.
9 @prefix tree: <http://www.w3.org/ns/shapetree#>.
10
11 <platform_0004a30b0024e96c.ttl> a sosa:Platform;
12     sosa:hosts <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_air_temperature>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_atmospheric_pressure>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_battery_voltage>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_compass_heading>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_east_wind_speed>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_average_distance>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_strike_count>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_maximum_wind_speed>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_north_wind_speed>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_precipitation>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_relative_humidity>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_sensor_temperature_internal>,
          <platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_solar_radiation>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_vapor_pressure>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_direction>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_x_orientation_angle>, <
          platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_y_orientation_angle>.
```

```
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_air_temperature> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:airTemperature.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_atmospheric_pressure> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:atmosphericPressure.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_battery_voltage> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:batteryVoltage.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_compass_heading> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:compassHeading.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_east_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:eastWindSpeed.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_average_distance> a sosa:
    Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:lightningAverageDistance.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_lightning_strike_count> a sosa:Sensor
    ;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:lightningStrikeCount.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_maximum_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:maximumWindSpeed.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_north_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:northWindSpeed.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_precipitation> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:precipitation.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_relative_humidity> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:relativeHumidity.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_sensor_temperature_internal> a sosa:
    Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:sensorTemperatureInternal.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_solar_radiation> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:solarRadiation.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_vapor_pressure> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
```

```
    sosa:observes obs:vaporPressure.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_direction> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:windDirection.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:windSpeed.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_x_orientation_angle> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:xOrientationAngle.
<platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_y_orientation_angle> a sosa:Sensor;
    sosa:isHostedBy <platform_0004a30b0024e96c.ttl>;
    sosa:observes obs:yOrientationAngle.
```

E.2 Root file

```
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix obs: <http://localhost:8080/shapes/ObservableProperties.ttl>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix tree: <http://www.w3.org/ns/shapetree#>.

<root_0004a30b0024e96c.ttl#Collection> a ldp:Container;
    <https://raw.githubusercontent.com/shapetrees/specification/master/shapetree.ttl#
        validatedBy> <http://localhost:8080/shapes/Sensor.shex#Observation>;
    <https://w3id.org/tree#view> <root_0004a30b0024e96c.ttl>.
<root_0004a30b0024e96c.ttl> a <https://w3id.org/tree#Node>;
    <https://w3id.org/tree#relation> _:b0.
_:b0 a <https://w3id.org/tree#GreaterThanOrEqualToRelation>;
    <https://w3id.org/tree#path> sosa:resultTime;
    <https://w3id.org/tree#node> <today_0004a30b0024e96c.ttl>;
    <https://w3id.org/tree#value> "2021-05-28T00:00:00.000Z"^^xsd:dateTime.
```

E.3 Members of the collection

```
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix sosa: <http://www.w3.org/ns/sosa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix obs: <http://localhost:8080/shapes/ObservableProperties.ttl>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix tree: <http://www.w3.org/ns/shapetree#>.

<today_0004a30b0024e96c.ttl#Collection> a ldp:Container;
    <https://raw.githubusercontent.com/shapetrees/specification/master/shapetree.ttl#
        validatedBy> <http://localhost:8080/shapes/Sensor.shex#Observation>;
    <https://w3id.org/tree#member> <sensor_0004a30b0024e96c_air_temperature_observation2294>,
         <sensor_0004a30b0024e96c_atmospheric_pressure_observation2294>, <
        sensor_0004a30b0024e96c_battery_voltage_observation2294>, <
        sensor_0004a30b0024e96c_compass_heading_observation2294>, <
        sensor_0004a30b0024e96c_east_wind_speed_observation2294>, <
        sensor_0004a30b0024e96c_lightning_average_distance_observation2294>, <
        sensor_0004a30b0024e96c_lightning_strike_count_observation2294>, <
        sensor_0004a30b0024e96c_maximum_wind_speed_observation2294>, <
        sensor_0004a30b0024e96c_north_wind_speed_observation2294>, <
        sensor_0004a30b0024e96c_precipitation_observation2294>, <
        sensor_0004a30b0024e96c_relative_humidity_observation2294>, <
        sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294>, <
        sensor_0004a30b0024e96c_solar_radiation_observation2294>, <
        sensor_0004a30b0024e96c_vapor_pressure_observation2294>, <
        sensor_0004a30b0024e96c_wind_direction_observation2294>, <
        sensor_0004a30b0024e96c_wind_speed_observation2294>, <
        sensor_0004a30b0024e96c_x_orientation_angle_observation2294>, <
        sensor_0004a30b0024e96c_y_orientation_angle_observation2294>.
<today_0004a30b0024e96c.ttl> a <https://w3id.org/tree#Node>.
<sensor_0004a30b0024e96c_air_temperature_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_air_temperature_observation_result2294>;
    sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_air_temperature>;
    sosa:observedProperty obs:airTemperature;
    sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
<sensor_0004a30b0024e96c_air_temperature_observation_result2294> a qudt:QuantityValue;
    qudt:Unit unit:DEG_C;
    qudt:numericValue "12.9"^^xsd:float;
    sosa:isResultOf <sensor_0004a30b0024e96c_air_temperature_observation2294>.
<sensor_0004a30b0024e96c_atmospheric_pressure_observation2294> a sosa:Observation;
    sosa:hasResult <sensor_0004a30b0024e96c_atmospheric_pressure_observation_result2294>;
    sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
        sensor_0004a30b0024e96c_atmospheric_pressure>;
    sosa:observedProperty obs:atmosphericPressure;
```

```
28      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
29  <sensor_0004a30b0024e96c_atmospheric_pressure_observation_result2294> a qudt:QuantityValue;
30      qudt:Unit unit:KiloPA;
31      qudt:numericValue "99.77"^^xsd:float;
32      sosa:isResultOf <sensor_0004a30b0024e96c_atmospheric_pressure_observation2294>.
33  <sensor_0004a30b0024e96c_battery_voltage_observation2294> a sosa:Observation;
34      sosa:hasResult <sensor_0004a30b0024e96c_battery_voltage_observation_result2294>;
35      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_battery_voltage>;
36      sosa:observedProperty obs:batteryVoltage;
37      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
38  <sensor_0004a30b0024e96c_battery_voltage_observation_result2294> a qudt:QuantityValue;
39      qudt:Unit unit:EV;
40      qudt:numericValue "2.661"^^xsd:float;
41      sosa:isResultOf <sensor_0004a30b0024e96c_battery_voltage_observation2294>.
42  <sensor_0004a30b0024e96c_compass_heading_observation2294> a sosa:Observation;
43      sosa:hasResult <sensor_0004a30b0024e96c_compass_heading_observation_result2294>;
44      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_compass_heading>;
45      sosa:observedProperty obs:compassHeading;
46      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
47  <sensor_0004a30b0024e96c_compass_heading_observation_result2294> a qudt:QuantityValue;
48      qudt:Unit unit:DEG;
49      qudt:numericValue "0"^^xsd:float;
50      sosa:isResultOf <sensor_0004a30b0024e96c_compass_heading_observation2294>.
51  <sensor_0004a30b0024e96c_east_wind_speed_observation2294> a sosa:Observation;
52      sosa:hasResult <sensor_0004a30b0024e96c_east_wind_speed_observation_result2294>;
53      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_east_wind_speed>;
54      sosa:observedProperty obs:eastWindSpeed;
55      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
56  <sensor_0004a30b0024e96c_east_wind_speed_observation_result2294> a qudt:QuantityValue;
57      qudt:Unit unit:M-PER-SEC;
58      qudt:numericValue "-1.05"^^xsd:float;
59      sosa:isResultOf <sensor_0004a30b0024e96c_east_wind_speed_observation2294>.
60  <sensor_0004a30b0024e96c_lightning_average_distance_observation2294> a sosa:Observation;
61      sosa:hasResult <sensor_0004a30b0024e96c_lightning_average_distance_observation_result2294
            >;
62      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_lightning_average_distance>;
63      sosa:observedProperty obs:lightningAverageDistance;
64      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
65  <sensor_0004a30b0024e96c_lightning_average_distance_observation_result2294> a qudt:
        QuantityValue;
```

```
66      qudt:Unit unit:KiloM;
67      qudt:numericValue "0"^^xsd:float;
68      sosa:isResultOf <sensor_0004a30b0024e96c_lightning_average_distance_observation2294>.
69  <sensor_0004a30b0024e96c_lightning_strike_count_observation2294> a sosa:Observation;
70      sosa:hasResult <sensor_0004a30b0024e96c_lightning_strike_count_observation_result2294>;
71      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_lightning_strike_count>;
72      sosa:observedProperty obs:lightningStrikeCount;
73      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
74  <sensor_0004a30b0024e96c_lightning_strike_count_observation_result2294> a qudt:QuantityValue
        ;
75      qudt:numericValue "0"^^xsd:float;
76      sosa:isResultOf <sensor_0004a30b0024e96c_lightning_strike_count_observation2294>.
77  <sensor_0004a30b0024e96c_maximum_wind_speed_observation2294> a sosa:Observation;
78      sosa:hasResult <sensor_0004a30b0024e96c_maximum_wind_speed_observation_result2294>;
79      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_maximum_wind_speed>;
80      sosa:observedProperty obs:maximumWindSpeed;
81      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
82  <sensor_0004a30b0024e96c_maximum_wind_speed_observation_result2294> a qudt:QuantityValue;
83      qudt:Unit unit:M-PER-SEC;
84      qudt:numericValue "3.08"^^xsd:float;
85      sosa:isResultOf <sensor_0004a30b0024e96c_maximum_wind_speed_observation2294>.
86  <sensor_0004a30b0024e96c_north_wind_speed_observation2294> a sosa:Observation;
87      sosa:hasResult <sensor_0004a30b0024e96c_north_wind_speed_observation_result2294>;
88      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_north_wind_speed>;
89      sosa:observedProperty obs:northWindSpeed;
90      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
91  <sensor_0004a30b0024e96c_north_wind_speed_observation_result2294> a qudt:QuantityValue;
92      qudt:Unit unit:M-PER-SEC;
93      qudt:numericValue "0.05"^^xsd:float;
94      sosa:isResultOf <sensor_0004a30b0024e96c_north_wind_speed_observation2294>.
95  <sensor_0004a30b0024e96c_precipitation_observation2294> a sosa:Observation;
96      sosa:hasResult <sensor_0004a30b0024e96c_precipitation_observation_result2294>;
97      sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_precipitation>;
98      sosa:observedProperty obs:precipitation;
99      sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
100 <sensor_0004a30b0024e96c_precipitation_observation_result2294> a qudt:QuantityValue;
101     qudt:Unit unit:MilliM;
102     qudt:numericValue "0"^^xsd:float;
103     sosa:isResultOf <sensor_0004a30b0024e96c_precipitation_observation2294>.
104 <sensor_0004a30b0024e96c_relative_humidity_observation2294> a sosa:Observation;
```

```
105     sosa:hasResult <sensor_0004a30b0024e96c_relative_humidity_observation_result2294>;
106     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_relative_humidity>;
107     sosa:observedProperty obs:relativeHumidity;
108     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
109 <sensor_0004a30b0024e96c_relative_humidity_observation_result2294> a qudt:QuantityValue;
110     qudt:Unit unit:PERCENT_RH;
111     qudt:numericValue "57.5"^^xsd:float;
112     sosa:isResultOf <sensor_0004a30b0024e96c_relative_humidity_observation2294>.
113 <sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294> a sosa:Observation;
114     sosa:hasResult <
            sensor_0004a30b0024e96c_sensor_temperature_internal_observation_result2294>;
115     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_sensor_temperature_internal>;
116     sosa:observedProperty obs:sensorTemperatureInternal;
117     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
118 <sensor_0004a30b0024e96c_sensor_temperature_internal_observation_result2294> a qudt:
        QuantityValue;
119     qudt:Unit unit:DEG_C;
120     qudt:numericValue "13.1"^^xsd:float;
121     sosa:isResultOf <sensor_0004a30b0024e96c_sensor_temperature_internal_observation2294>.
122 <sensor_0004a30b0024e96c_solar_radiation_observation2294> a sosa:Observation;
123     sosa:hasResult <sensor_0004a30b0024e96c_solar_radiation_observation_result2294>;
124     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_solar_radiation>;
125     sosa:observedProperty obs:solarRadiation;
126     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
127 <sensor_0004a30b0024e96c_solar_radiation_observation_result2294> a qudt:QuantityValue;
128     qudt:Unit unit:W-PER-M2;
129     qudt:numericValue "19"^^xsd:float;
130     sosa:isResultOf <sensor_0004a30b0024e96c_solar_radiation_observation2294>.
131 <sensor_0004a30b0024e96c_vapor_pressure_observation2294> a sosa:Observation;
132     sosa:hasResult <sensor_0004a30b0024e96c_vapor_pressure_observation_result2294>;
133     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_vapor_pressure>;
134     sosa:observedProperty obs:vaporPressure;
135     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
136 <sensor_0004a30b0024e96c_vapor_pressure_observation_result2294> a qudt:QuantityValue;
137     qudt:Unit unit:KiloPA;
138     qudt:numericValue "0.86"^^xsd:float;
139     sosa:isResultOf <sensor_0004a30b0024e96c_vapor_pressure_observation2294>.
140 <sensor_0004a30b0024e96c_wind_direction_observation2294> a sosa:Observation;
141     sosa:hasResult <sensor_0004a30b0024e96c_wind_direction_observation_result2294>;
```

```
142     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_wind_direction>;
143     sosa:observedProperty obs:windDirection;
144     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
145 <sensor_0004a30b0024e96c_wind_direction_observation_result2294> a qudt:QuantityValue;
146     qudt:Unit unit:DEG;
147     qudt:numericValue "272.6"^^xsd:float;
148     sosa:isResultOf <sensor_0004a30b0024e96c_wind_direction_observation2294>.
149 <sensor_0004a30b0024e96c_wind_speed_observation2294> a sosa:Observation;
150     sosa:hasResult <sensor_0004a30b0024e96c_wind_speed_observation_result2294>;
151     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#sensor_0004a30b0024e96c_wind_speed>;
152     sosa:observedProperty obs:windSpeed;
153     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
154 <sensor_0004a30b0024e96c_wind_speed_observation_result2294> a qudt:QuantityValue;
155     qudt:Unit unit:M-PER-SEC;
156     qudt:numericValue "1.05"^^xsd:float;
157     sosa:isResultOf <sensor_0004a30b0024e96c_wind_speed_observation2294>.
158 <sensor_0004a30b0024e96c_x_orientation_angle_observation2294> a sosa:Observation;
159     sosa:hasResult <sensor_0004a30b0024e96c_x_orientation_angle_observation_result2294>;
160     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_x_orientation_angle>;
161     sosa:observedProperty obs:xOrientationAngle;
162     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
163 <sensor_0004a30b0024e96c_x_orientation_angle_observation_result2294> a qudt:QuantityValue;
164     qudt:Unit unit:DEG;
165     qudt:numericValue "2.4"^^xsd:float;
166     sosa:isResultOf <sensor_0004a30b0024e96c_x_orientation_angle_observation2294>.
167 <sensor_0004a30b0024e96c_y_orientation_angle_observation2294> a sosa:Observation;
168     sosa:hasResult <sensor_0004a30b0024e96c_y_orientation_angle_observation_result2294>;
169     sosa:madeBySensor <../platform_0004a30b0024e96c.ttl#
            sensor_0004a30b0024e96c_y_orientation_angle>;
170     sosa:observedProperty obs:yOrientationAngle;
171     sosa:resultTime "2021-05-28T16:31:08.904Z"^^xsd:dateTime.
172 <sensor_0004a30b0024e96c_y_orientation_angle_observation_result2294> a qudt:QuantityValue;
173     qudt:Unit unit:DEG;
174     qudt:numericValue "-3.1"^^xsd:float;
175     sosa:isResultOf <sensor_0004a30b0024e96c_y_orientation_angle_observation2294>.
```

Appendix F - RML file

The following mapping file is used to translate the intermediate representation from Appendix C to a SOSA RDF representation like the Turtle resource from Appendix D.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qudt: <http://qudt.org/schema/qudt/>.
@prefix unit: <http://qudt.org/vocab/unit/>.
@prefix obs: <http://localhost:8080/shapes/ObservableProperties.ttl>.
@base <-->.

<#ResultMapping>
    rml:logicalSource [
        rml:source "input";
        rml:iterator "$.measurements.[*]";
        rml:referenceFormulation ql:JSONPath
    ];

    rr:subjectMap [
        rr:template "http://localhost/sensor_{uuid}_{property}_observation_result{observation
            .id}";
        rr:class qudt:QuantityValue
    ];

    rr:predicateObjectMap [
        rr:predicate qudt:Unit;
        rr:objectMap [
            rr:template "http://qudt.org/vocab/unit/{observation.unit}"
        ]
    ];

    rr:predicateObjectMap [
        rr:predicate qudt:numericValue;
        rr:objectMap [
            rr:reference "observation.value";
            rr:datatype xsd:float
        ]
    ];
```

```
39
40      rr:predicateObjectMap [
41          rr:predicate sosa:isResultOf;
42          rr:objectMap [
43              rr:parentTriplesMap <#ObservationMapping>;
44              rr:joinCondition [
45                   rr:child "property";
46                   rr:parent "property";
47              ];
48          ]
49      ].
50
51  <#ObservationMapping>
52      rml:logicalSource [
53          rml:source "input";
54          rml:iterator "$.measurements[*]";
55          rml:referenceFormulation ql:JSONPath
56      ];
57
58      rr:subjectMap [
59          rr:template "http://localhost/sensor_{uuid}_{property}_observation{observation.id}";
60          rr:class sosa:Observation
61      ];
62
63      rr:predicateObjectMap [
64          rr:predicate sosa:resultTime;
65          rr:objectMap [
66              rr:reference "observation.time";
67              rr:datatype xsd:dateTime
68          ];
69      ];
70
71      rr:predicateObjectMap [
72          rr:predicate sosa:observedProperty;
73          rr:objectMap [
74              rr:template "http://localhost:8080/shapes/ObservableProperties.ttl{observation.
                    observableProperty}"
75          ];
76      ];
77
78      rr:predicateObjectMap [
79          rr:predicate sosa:madeBySensor;
80          rr:objectMap [
81              rr:parentTriplesMap <#SensorMapping>;
```

```
82            rr:joinCondition [
83                rr:child "property";
84                rr:parent "property";
85            ];
86        ]
87    ];
88
89    rr:predicateObjectMap [
90        rr:predicate sosa:hasResult;
91        rr:objectMap [
92            rr:parentTriplesMap <#ResultMapping>;
93            rr:joinCondition [
94                rr:child "property";
95                rr:parent "property";
96            ];
97        ]
98    ].
99
100 <#SensorMapping>
101     rml:logicalSource [
102         rml:source "input";
103         rml:iterator "$.measurements[*]";
104         rml:referenceFormulation ql:JSONPath
105     ];
106
107     rr:subjectMap [
108         rr:template "http://localhost/platform_{uuid}.ttl#sensor_{uuid}_{property}";
109         rr:class sosa:Sensor
110     ];
111
112
113     rr:predicateObjectMap [
114         rr:predicate sosa:observes;
115         rr:objectMap [
116             rr:template "http://localhost:8080/shapes/ObservableProperties.ttl{observation.
                observableProperty}"
117         ]
118     ];
119
120     rr:predicateObjectMap [
121         rr:predicate sosa:isHostedBy;
122         rr:objectMap [
123             rr:parentTriplesMap <#DeviceMapping>;
124         ]
```

```
125     ];
126
127     rr:predicateObjectMap [
128         rr:predicate sosa:madeObservation;
129         rr:objectMap [
130             rr:template "http://localhost/sensor_{uuid}_{property}_observation{observation.id
                    }";
131         ];
132     ].
133
134 <#DeviceMapping>
135     rml:logicalSource [
136         rml:source "input";
137         rml:iterator "$";
138         rml:referenceFormulation ql:JSONPath
139     ];
140
141     rr:subjectMap [
142         rr:template "http://localhost/platform_{uuid}.ttl";
143         rr:class sosa:Platform
144     ];
145
146     rr:predicateObjectMap [
147         rr:predicate sosa:hosts;
148         rr:objectMap [
149             rr:parentTriplesMap <#SensorMapping>;
150         ]
151     ].
```

Appendix G - Shape Expressions file

The file below contains the Shape Expressions file used both for the plain SOSA model as the TREE collection using SOSA model.

For the plain version, the shapes concerning the TREE syntax are not used.

```
PREFIX ldp: <http://www.w3.org/ns/ldp#>
PREFIX tree: <https://w3id.org/tree#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX st: <http://www.w3.org/ns/st#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX qudt: <http://qudt.org/schema/qudt/>
PREFIX unit: <http://qudt.org/vocab/unit/>
PREFIX obs: <http://localhost:8080/shapes/ObservableProperties.ttl>
PREFIX ldes: <https://w3id.org/ldes#>

<#Container> {
    a [ldp:BasicContainer]
}
<#Collection> {
    tree:member @<#Observation> *
} AND {
    ( a [ldp:Container] ;
    st:validatedBy IRI) |
    ( a [ldes:EventStream tree:Collection] ;
    tree:shape IRI)
}
<#CollectionRoot> @<#Collection> AND {
    tree:view @<#Node>
}
<#Node> {
    a [tree:Node];
    tree:relation @<#Relation> *
}
<#Relation> {
 a [tree:GreaterThanOrEqualToRelation];
 tree:path [sosa:resultTime] ;
 tree:node @<#Node> ;
 tree:value xsd:dateTime
}
<#Platform> {
```

```
38     a [sosa:Platform] ;
39     rdfs:label xsd:string OR rdf:langString?;
40     rdfs:comment xsd:string OR rdf:langString ?;
41     sosa:hosts @<#Sensor> +
42 }
43 <#Sensor> {
44     a [sosa:Sensor] ;
45     rdfs:label xsd:string ? ;
46     rdfs:comment xsd:string OR rdf:langString ? ; } AND {
47     ( sosa:madeObservation @<#AirTemperatureObservation> * ;
48     sosa:observes [obs:airTemperature]) |
49     ( sosa:madeObservation @<#AtmosphericPressureObservation> * ;
50     sosa:observes [obs:atmosphericPressure]) |
51     ( sosa:madeObservation @<#BatteryVoltageObservation> * ;
52     sosa:observes [obs:batteryVoltage]) |
53     ( sosa:madeObservation @<#CompassHeadingObservation> * ;
54     sosa:observes [obs:compassHeading]) |
55     ( sosa:madeObservation @<#EastWindSpeedObservation> * ;
56     sosa:observes [obs:eastWindSpeed]) |
57     ( sosa:madeObservation @<#LightningAverageDistanceObservation> * ;
58     sosa:observes [obs:lightningAverageDistance]) |
59     ( sosa:madeObservation @<#LightningStrikeCountObservation> * ;
60     sosa:observes [obs:lightningStrikeCount]) |
61     ( sosa:madeObservation @<#MaximumWindSpeedObservation> * ;
62     sosa:observes [obs:maximumWindSpeed]) |
63     ( sosa:madeObservation @<#NorthWindSpeedObservation> * ;
64     sosa:observes [obs:northWindSpeed]) |
65     ( sosa:madeObservation @<#PrecipitationObservation> * ;
66     sosa:observes [obs:precipitation]) |
67     ( sosa:madeObservation @<#RelativeHumidityObservation> * ;
68     sosa:observes [obs:relativeHumidity]) |
69     ( sosa:madeObservation @<#SensorTemperatureInternalObservation> * ;
70     sosa:observes [obs:sensorTemperatureInternal]) |
71     ( sosa:madeObservation @<#SolarRadiationObservation> * ;
72     sosa:observes [obs:solarRadiation]) |
73     ( sosa:madeObservation @<#VaporPressureObservation> * ;
74     sosa:observes [obs:vaporPressure]) |
75     ( sosa:madeObservation @<#WindDirectionObservation> * ;
76     sosa:observes [obs:windDirection]) |
77     ( sosa:madeObservation @<#WindSpeedObservation> * ;
78     sosa:observes [obs:windSpeed]) |
79     ( sosa:madeObservation @<#XOrientationAngleObservation> * ;
80     sosa:observes [obs:xOrientationAngle]) |
81     ( sosa:madeObservation @<#YOrientationAngleObservation> * ;
```

```
82     sosa:observes [obs:yOrientationAngle])
83 }
84 <#Observation> {
85     a [sosa:Observation] ;
86     sosa:resultTime xsd:dateTime ;
87     rdfs:comment xsd:string OR rdf:langString ? ;
88     rdfs:label xsd:string OR rdf:langString ?;
89     sosa:madeBySensor @<#Sensor> ?
90 }
91 <#AirTemperatureObservation> @<#Observation> AND {
92     sosa:hasResult @<#TemperatureResult> ;
93     sosa:observedProperty [obs:airTemperature] ?
94 }
95 <#TemperatureResult> {
96     a [qudt:QuantityValue];
97     qudt:numericValue xsd:float ;
98     qudt:Unit [unit:DEG_C]
99 }
100 <#AtmosphericPressureObservation> @<#Observation> AND {
101     sosa:hasResult @<#PressureResult> ;
102     sosa:observedProperty [obs:atmosphericPressure] ?
103 }
104 <#PressureResult> {
105     a [qudt:QuantityValue];
106     qudt:numericValue xsd:float ;
107     qudt:Unit [unit:KiloPA ]
108 }
109 <#BatteryVoltageObservation> @<#Observation> AND {
110     sosa:hasResult @<#BatteryVoltageResult> ;
111     sosa:observedProperty [obs:batteryVoltage] ?
112 }
113 <#BatteryVoltageResult> {
114     a [qudt:QuantityValue];
115     qudt:numericValue xsd:float ;
116     qudt:Unit [unit:EV ]
117 }
118 <#CompassHeadingObservation> @<#Observation> AND {
119     sosa:hasResult @<#AngleResult> ;
120     sosa:observedProperty [obs:compassHeading] ?
121 }
122 <#AngleResult> {
123     a [qudt:QuantityValue];
124     qudt:numericValue xsd:float ;
125     qudt:Unit [unit:DEG ]
```

```
126 }
127 <#EastWindSpeedObservation> @<#Observation> AND {
128     sosa:hasResult @<#SpeedResult> ;
129     sosa:observedProperty [obs:eastWindSpeed] ?
130 }
131 <#SpeedResult> {
132     a [qudt:QuantityValue];
133     qudt:numericValue xsd:float ;
134     qudt:Unit [unit:M-PER-SEC ]
135 }
136 <#LightningAverageDistanceObservation> @<#Observation> AND {
137     sosa:hasResult @<#DistanceResult> ;
138     sosa:observedProperty [obs:lightningAverageDistance] ?
139 }
140 <#DistanceResult> {
141     a [qudt:QuantityValue];
142     qudt:numericValue xsd:float ;
143     qudt:Unit [unit:KiloM ]
144 }
145 <#LightningStrikeCountObservation> @<#Observation> AND {
146     sosa:hasResult @<#CountResult> ;
147     sosa:observedProperty [obs:lightningStrikeCount] ?
148 }
149 <#CountResult> {
150     a [qudt:QuantityValue];
151     qudt:numericValue xsd:float
152 }
153 <#MaximumWindSpeedObservation> @<#Observation> AND {
154     sosa:hasResult @<#SpeedResult> ;
155     sosa:observedProperty [obs:maximumWindSpeed] ?
156 }
157 <#NorthWindSpeedObservation> @<#Observation> AND {
158     sosa:hasResult @<#SpeedResult> ;
159     sosa:observedProperty [obs:northWindSpeed] ?
160 }
161 <#PrecipitationObservation> @<#Observation> AND {
162     sosa:hasResult @<#PrecipitationResult> ;
163     sosa:observedProperty [obs:precipitation] ?
164 }
165 <#PrecipitationResult> {
166     a [qudt:QuantityValue];
167     qudt:numericValue xsd:float ;
168     qudt:Unit [unit:MilliM ]
169 }
```

```
<#RelativeHumidityObservation> @<#Observation> AND {
    sosa:hasResult @<#HumidityResult> ;
    sosa:observedProperty [obs:relativeHumidity] ?
}
<#HumidityResult> {
    a [qudt:QuantityValue];
    qudt:numericValue xsd:float ;
    qudt:Unit [unit:PERCENT_RH ]
}
<#SensorTemperatureInternalObservation> @<#Observation> AND {
    sosa:hasResult @<#TemperatureResult> ;
    sosa:observedProperty [obs:sensorTemperatureInternal] ?
}
<#SolarRadiationObservation> @<#Observation> AND {
    sosa:hasResult @<#IrradianceResult> ;
    sosa:observedProperty [obs:solarRadiation] ?
}
<#IrradianceResult> {
    a [qudt:QuantityValue];
    qudt:numericValue xsd:float ;
    qudt:Unit [unit:W-PER-M2 ]
}
<#VaporPressureObservation> @<#Observation> AND {
    sosa:hasResult @<#PressureResult> ;
    sosa:observedProperty [obs:vaporPressure] ?
}
<#WindDirectionObservation> @<#Observation> AND {
    sosa:hasResult @<#AngleResult> ;
    sosa:observedProperty [obs:windDirection] ?
}
<#WindSpeedObservation> @<#Observation> AND {
    sosa:hasResult @<#SpeedResult> ;
    sosa:observedProperty [obs:windSpeed] ?
}
<#XOrientationAngleObservation> @<#Observation> AND {
    sosa:hasResult @<#AngleResult> ;
    sosa:observedProperty [obs:xOrientationAngle] ?
}
<#YOrientationAngleObservation> @<#Observation> AND {
    sosa:hasResult @<#AngleResult> ;
    sosa:observedProperty [obs:yOrientationAngle] ?
}
```

Appendix H - Shape Trees

File 1 is the Shape tree file for the SOSA model. File 2 is the Shape tree file which provides validation to form the structure shown in 3.5, indiciting that is for the TREE collection using SOSA model.

H.1 Shape Tree for the SOSA model

```
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix st: <http://www.w3.org/ns/shapetree#> .

<#container> st:expectsType ldp:Container ;
  st:contains <#platform> .
<#platform> st:expectsType ldp:Resource ;
  st:matchesUriTemplate "{platformName}" ;
  st:validatedBy <Sensor.shex#Platform> .
```

H.2 Shape Tree for the TREE collection using SOSA model

```
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix st: <http://www.w3.org/ns/shapetree#> .

<#container> st:expectsType ldp:Container ;
  st:matchesUriTemplate "platform{}" ;
  st:contains <#collection>, <#platform> .
<#collection> st:expectsType ldp:Container ;
  st:matchesUriTemplate "Collections/" ;
  st:validatedBy <Collection.shex#Container> ;
    st:contains <#root_node>, <#today_node>, <#last_7_days_node> , <#current_month>, <#
        archive_node>.
<#platform> st:expectsType ldp:Resource ;
  st:matchesUriTemplate "platform_{platformName}.ttl" ;
  st:validatedBy <Collection.shex#Platform> .
<#root_node> st:expectsType ldp:Resource ;
    st:matchesUriTemplate "root_{uuid}.ttl";
    st:validatedBy <Collection.shex#CollectionRoot> .
<#today_node> st:expectsType ldp:Resource ;
    st:matchesUriTemplate "today_{uuid}.ttl";
    st:validatedBy <Collection.shex#Collection> .
<#last_7_days_node> st:expectsType ldp:Resource ;
    st:matchesUriTemplate "last_7_days_{uuid}.ttl";
```

```
22     st:validatedBy <Collection.shex#Collection> .
23 <#current_month> st:expectsType ldp:Resource ;
24     st:matchesUriTemplate "current_month_{uuid}.ttl";
25     st:validatedBy <Collection.shex#Collection> .
26 <#archive_node> st:expectsType ldp:Resource ;
27     st:matchesUriTemplate "2{year}_{month}_{uuid}.ttl";
28     st:validatedBy <Collection.shex#Collection> .
```