Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Doan Kien Bui

Design tools for ontology-based network communication protocols

Master's Thesis
Espoo, August 28, 2019

Supervisor: Professor Petri Vuorimaa, Aalto University
Advisor: Jani Hursti PhD. (Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Doan Kien Bui

Title:
Design tools for ontology-based network communication protocols

Date: August 28, 2019 **Pages:** 61

Major: Computer Science **Code:** SCI3042

Supervisor: Professor Petri Vuorimaa

Advisor: Jani Hursti PhD. (Tech.)

Internet of Things has evolved quickly and reached to every aspect of our lives over the years. The number of new heterogeneous, distributed devices and applications connecting to the Internet is growing exponentially every day. As a result, data interoperability has become a prerequisite for IoT networks.

However, the current infrastructures and communication protocols do not provide a convenient way for applications from different domains to interpret and process each other's data, which is stored in vastly diversified, non-standardized formats. Due to this lack of common ground, in many cases, the integration overhead hinders organisations from exchanging their data to generate business values. Semantic technologies would be a promising solution for these issues, thanks to its ability to capture the high-level meaning of data.

Asema is developing SmartAPI, a semantics-based API framework for sharing data between IoT systems. This thesis work is a part of SmartAPI project, focuses on designing and developing a data designer application. I build a single page web application with a modern graphical user interface, allowing users to create, organise and share data models.

Keywords: IoT, semantic web, ontology, data interoperability, user interface, user experience

Language: English

Acknowledgements

First and foremost, I would like to express my profound gratitude to my supervisor Professor Petri Vuorimaa and my advisor Jani Hursti. I would not finish this thesis work without for their incredible guidance and support.

Secondly, I would like to thank all members of Asema Oy and Professor Petri's research group. It has been an enjoyable working experience with you all.

Last but not least, I would like to thank my friends and family for their valuable support.

Thank you all for everything.

Espoo, August 28, 2019

Doan Kien Bui

Abbreviations and Acronyms

IoT	Internet of Things
API	Application Program Interface
SDK	Software Development Kit
RFID	Radio Frequency Identification
WSN	Wireless Sensor Networks
M2M	Machine to Machine
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
OM2M	Open Machine to Machine
HTTP	Hypertext Transfer Protocol
CoAP	Constrained Application Protocol
REST	Representational State Transfer
RDF	Resource Description Framework
IRI	Internationalized Resource Identifier
URI	Unique Resource Identifier
XML	Extensible Markup Language
SKOS	Simple Knowledge Organization System
OWL	Web Ontology Language
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
ERP	Enterprise Resource Planning
SSO	Single sign-on
HTML	Hypertext Markup Language
CSS	Cascading style sheets
AJAX	Asynchronous JavaScript and XML

Contents

Chapter 1

Introduction

Since its birth in the 1960s [1], the Internet has revolutionised various industries and completely changed the way information technology involves in everyday human activities. Currently connecting 4.3 billion users over the globe, and growing at over a million new users each day [2], the Internet has been woven into the fabric of many lives on the planet. The Internet connectivity is empowering an enormous spectrum of applications and devices, visibly influencing the way we communicate, the way we work and the way we live. It can be found everywhere, from workstations to personal computers, from smart homes, smart household appliances to wearable devices and smart sensors. And that global network of connected devices is called the Internet of Things (IoT).

Over recent years, IoT and its applications have been the topic of interest for both academic and industrial players [3]. IoT networks consist of interconnected smart devices, which communicate with each other autonomously without human involvement. IoT devices are heterogeneous and diversified in infrastructures, interfaces, and processing and sensoring capabilities. Although they are often limited in power, memory and computing power, the communication processes among IoT devices can generate an enormous amount of data. Therefore, data interoperability has become a challenging issue in the IoT world [4].

Many studies have been conducted on applying semantic technology to solve the IoT data interoperability problem [5]. Semantic Web refers to machine-readable data on the Internet. Not only describing data in machine-consumable formats, Semantic Web also specifies the meaning of data and its relations. This enables data to be processed and stored in a self-explanatory format so that different domain applications can share and interpret each other's data unambiguously.

1.1 Problem statement

It is often that to extract significant business values from IoT data, multiple sources of data from different organisations need to be combined [6]. However, due to IoT heterogeneous characteristics, raw IoT data is usually processed and stored in various formats with non-standard naming and vocabulary conventions. Subsequently, the industry has customarily suffered from lacking a consensus of data interoperability. Semantic technology has shown potential as a promising solution because it is not only able to present data in standardised, machine-readable formats but also able to describe what the data stands for.

This thesis is a part of Smart API, an open-source project developed and maintained by Asema Oy, designed for semantic interoperability of IoT systems. It helps two or more IoT applications to understand, analyse and interpret each other's data effectively and unambiguously. However, in a network of systems that includes different domain applications, creating, editing and sharing data models is challenging. Therefore, in the scope of this thesis, the main research question we try to answer is: **How to create and share ontology-based data models to improve the interoperability and compatibility in IoT systems**.

1.2 Scope and Goals

The thesis focuses on designing and implementing a graphical user interface for creating, editing and sharing triples. Users can use the tool for browsing existing data models, creating new ones, and generating code stubs to use with the Smart API SDK and Smart API services. The application must require minimal installation effort so that it can be integrated easily with any infrastructure. The thesis has two primary objectives:

- **Improving the overall usability**: Users should be able to use all functionalities in the graphical user interface effortlessly and intuitively
- **Implementing multi-dimensional data models**: We target to design and implement the support for more complex, multi-dimensional data models

The new data designer tool will be a part of Smart API services in Asema IoT Solution.

1.3 Structure of the Thesis

The thesis comprises six chapters in the following order: Introduction, Background, Methodology, Design and Implementation, Evaluation and Conclusion. Chapter 2 gives an overview of the basis and current research about Internet of Things, data interoperability and Semantic Web. Chapter 3 explains the research methodology applied in this thesis work. Chapter 4 describes the design and implementation of the proposed solution, which is a web-based ontology editing application. Chapter 5 shows the evaluation of the proposed solution in a practical use case and benchmarks it with other existing ontology editors. Lastly, Chapter 6 contains the conclusion, retrospect, and directions for future works.

Chapter 2

Background

In this chapter, I will review the technologies and literature that are related to this thesis work. I first thoroughly go through the concepts and principles of the Internet of Things to set a base for the thesis. Then, the current state-of-the-art of data exchange and data management in the Internet of Things industry is discussed. Finally, I inspect the Semantic Web and Smart API as a part of the Internet of Things interoperability architecture.

2.1 Internet of things

Internet of Things is referred to as systems of smart devices interconnected via a linked network [7]. There are two pillars in IoT: "Internet" and "Things". "Things" are physical, smart devices that function autonomously and are able to communicate with each other or with a server via Internet connectivity. This includes everything from small-sized, individual wearable devices such as sensors, fitness wristbands and smart watches, to household appliances, such as smart television, smart refrigerators, washing machines, smart home systems, to large-scale, distributed systems such as traffic lights, aeroplane engines and IoT Gateways [8]. IoT applications have covered a wide range of industries, from manufacturing [9], logistics [10], retailing [11], to pharmaceutics and healthcare services [12].

2.1.1 Overview

Internet of Things and its applications are transforming business processes by enabling machines and devices to interact with each other automatically and intelligently, improving the visibility, accuracy and real-time access of data flow. IoT envisions a new world where a global network of connected devices

significantly enhances human life quality [13]. To realise IoT visions, new characteristics and requirements are introduced to the preexisting tradition computing devices [14]:

- **Ubiquitous Connectivity**: Perpetual connectivity is a fundamental principle in Internet of Things systems. Smart devices are required to be accessible without any time and place restrictions
- **Heterogeneity**: IoT devices diverse in a wide range of hardware and software. An IoT application needs to support a varied set of devices and connectivity protocols
- **Smartness**: IoT smartness consists of object smartness and network smartness. IoT devices need standardisation of communication standards, from device hardware layers that interact with the physical world, to software layers that handle the communication with the Internet.
- **Object Addressability and Functionality**: IoT devices are designed to be identified effortlessly and readily available for other purposes.
- **Resource Constraints**: IoT network architecture considers computing and power limitation of sensors and devices.

To establish the horizontal interaction between IoT entities, devices and sensors connected to the network constantly send data to the cloud, from which computer systems, software developers and other parties read, interact and integrate the data with their applications [15]. Figure 2.1 demonstrates a common network setup among different parties in an IoT communication network.

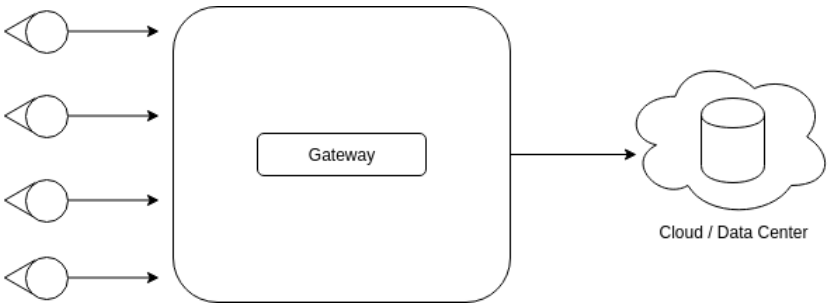

Figure 2.1: Communications in IoT networks.

There are five essential IoT technologies that are used in a wide range of IoT products and services [16]:

Radio Frequency Identification (RFID) refers to an IoT electromagnetic technology used to identify and trace the information that is attached to a digital data encoded tag [17]. Similar to traditional barcoding tag technology, information about the object is stored in a microchip on an RFID tag or a smart label. However, RFID offers numerous technological advantages, including the ability to read data out of line-of-sight. This allows RFID tags to be able to work hundreds of meters away from the tag reader, while traditional barcode tags have to be aligned with the optical scanner. RFID tags are widely used in manufacturing, hospital, asset management and retailing industries.

Wireless Sensor Networks (WSN) contains spatially disbursed, self-configured networks of autonomous dedicated sensor devices for tracking, monitoring, and storing the physical and environmental parameters [18]. WSN is used to measure the status of things, such as location, temperature, movements, and surrounding environmental conditions. Data in WSN is usually recorded and stored at a centralised location. A WSN deployment can consist of hundreds of thousands of sensor nodes. WSN supports multiple network and communication protocols, and WSN topologies range from simple network setup to multiple wireless communication. WSN applications cover a broad spectrum, from military applications to monitoring, transportation, agriculture and health care services.

Middleware is another major application area of IoT technology. IoT Middleware is an intermediate software layer between software applications to support data communication and input, output. It works as a software interface that abstracts away the details of different complex underlying technologies used in different applications so that they can be connected. The ability to free software developers from the mundane communication details between software is highly valuable for IoT systems. Middleware is an essential part of the IoT architecture for empowering connectivity of a huge number of heterogeneous sensors and devices [19]. Most middleware-based IoT architecture is service-oriented and supports dynamic network topology.

Cloud Computing is a convergence of technologies that enables on-demand uses of shared configurable computing resources. It enables organisations to quickly start new computer services and redistribute provisioned resources upon business changes. Infrastructure as a Service (IaaS), Platform as a

Service (PaaS) and Software as a Service (SaaS) are the three major service models in the cloud computing industry. Cloud computing offers a stable, powerful platform with high availability and high scalability for IoT applications. Thanks to the advances of Cloud Computing, the massive amount generated from IoT applications can be processed and stored safely and performantly [20].

IoT application software is what truly brings values to IoT-empowered enterprises [21]. Although raw IoT data prove to be hugely beneficial for enterprises, the true value of the IoT technology can only be fully achieved when raw IoT data is integrated with industry applications, such as inventory systems, customer support systems, business intelligence and analytics applications.

However, because of the diffusion of applications and smart devices, whose interfaces are not specified or standardised, data compatibility and interoperability has become one major problem in IoT networks [22]. The architecture of the data processing layer in IoT software systems is not designed to fully handle the heterogeneity and massive volume of IoT sensor and device data [23]. This thesis focuses on improving data compatibility and operability for IoT application software. In the following section, I review the current state of the art of the IoT data interoperability.

2.1.2 IoT Interoperability

As the broadband Internet has become more widely available, the cost of connecting is decreasing, more devices are being created with Internet-connection capabilities and sensors built into them, Internet of Things is evolving quickly, the number of IoT units is anticipated to exceed 26 billion units by 2020 [24]. With an increasing amount of sensors and devices ready to be interconnected, creating intelligent and autonomous IoT systems that allow participant devices to exchange data securely and inter-operate effortlessly has emerged as an upcoming challenge. One step towards solving this problem is the global adoption of universal standards and cost-effective solutions. This has been a topic of interest for both industrial players and academia. I discuss some of the research on this topic below.

Machine to Machine (M2M) refers to the machine data communication technology that allows devices or machines of the same type to communicate directly via wired or wireless connections, without human assistance. It is a broad term and not particularly applied to any kinds of networking or com-

munication protocols. M2M has been used in a wide range of applications, and it is considered an essential part of the Internet of Things [25]. However, M2M suffers from lacking a standard for data compatibility. One notable project tackling this issue is Open Machine to Machine.

Open Machine to Machine (OM2M) proposed by The European Telecommunications Standards Institute (ETSI), an ETSI-M2M compliant service platform for M2M interoperability in 2014 [26]. OM2M has a modular architecture and supports multiple protocol bindings such as HTTP and CoAP. The main feature of OM2M includes a RESTful API with open interfaces to enable developing services and applications independently of the underlying network.

AllJoyn is an open-source framework created and maintained by the AllSeen Alliance and Linux Foundation for enhancing the interoperability among Internet of Things devices and applications [27]. AllJoyn offers a universal, secure and programmable software connectivity framework and an open-source SDK that supports basic functionalities as discovery, connection management, message routing and security. It runs on major platforms, including Linux, Android, iOS, and Windows, and many other lightweight real-time operating systems.

IoTivity Project is another open-source framework for device-to-device software connectivity [27]. It was merged with AllJoyn in October 2016 [28]. The primary goal of IoTivity is to create an extensible and robust network architecture for smart and thin devices. The IoTivity offers a resource-based, RESTful API connectivity framework that is available in several languages.

INTER-IoT is an on-going project funded by the European Commission, aiming to improve the interoperability among heterogeneous Internet of Things platforms [29]. INTER-IoT offers an open layer-based framework and associated methodology for seamless integration of different IoT architectures.

To overcome the lack of a consensus standard, at Asema, I propose the Smart API, a Semantic Web-based data modelling solution for IoT data exchange via a standardised API framework. I discuss the Semantic Web in the next section.

2.2 Semantic Web

The semantic web is a World Wide Web extension, developed through World Wide Web Consortium (W3C) [30]. The primary purpose of Semantic Web is to establish a universal framework to create machine-readable content on the Internet, that enables data sharing and data reusability across different parties.

The term Semantic Web was initiated by Tim Berners Lee, who envisions the Semantic Web to be a web of machine-readable data so that computers can interpret and analyse all the content on the internet, including links, texts and transactions between people and computers [30]. Collections of open interconnected datasets can be referred to as Linked Data. Tim Berners Lee described Linked Open Data through four principles [31]:

1. Use URIs as names to identify things
2. Use HTTP URIs so that people can look up and access those names.
3. Provide useful information about resources using open standard formats, such as RDF, SPARQL
4. Include links to others URIS when publishing to the Internet, so that people can discover more.

He also suggested a five star deployment model for Open data [32]:

- **1-star**: Make data available on the web, in whatever format and under an open license
- **2-star**: Make data available on the web in structured, machine-readable format.
- **3-star**: Make data available on the web in a non-proprietary open format
- **4-star**: Use URIs to name resources
- **5-star**: Provide links to other data to promote content

Metadata and ontology are the essential foundations of Linked Data. While metadata provides a description language for web resources, ontology is an explicit specification of abstraction, that describes concepts and relations in a given domain.

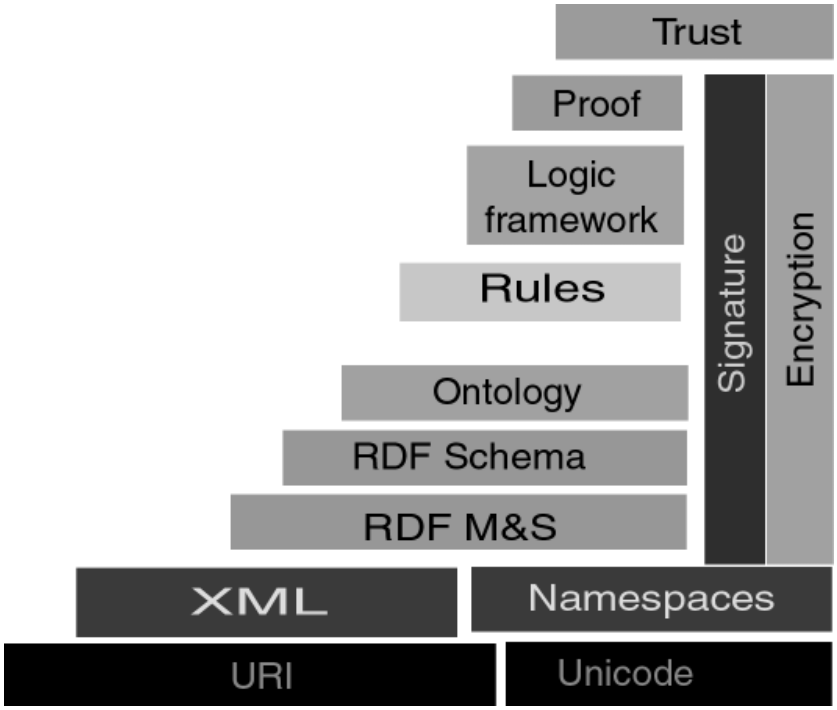

Figure 2.2: Tim Berners-Lee's Linked Data Layer Model [33]

To achieve and create Linked Data, data resources are given unique identifiers URIs [34]. By accessing the URIs, consumers can identify the given resource, find more information and related entities. URIs have been extended to IRIs that allow international characters [35]. Besides being identifiable and discoverable using IRIs, it's crucial that data is available in a common format. W3C introduces the Resource Description Framework (RDF) as a part of W3C specifications [30] in 1999 for metadata data modelling. An RDF statement can be visualised as an edge in a directed graph, which consists of a subject, an object and a predicate. Subject and object represent the defined resource and the target value. They can be an IRI, a blank node or a literal. A predicate is in the form of IRIs and represents the relationship or property. RDF has many syntax notations and data serialisation format. RDF statements can be serialized in XML based syntax (RDF/XML), triple notations (N3, Turtle, N-triples), or JSON format (JSON-LD) [30].

```
@prefix ad: <http://smart-api.io/ontology/ad#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ad:Tray1 rdf:type ad:Tray.
ad:Tray2 rdf:type ad:Tray.
```

The above example demonstrates simple RDF statements in Turtle format. RDF is very useful for describing relations between resources, yet it also has semation limitations. The power of RDF is further extended by RDFS, which provides a data-modelling mechanism for representing RDF vocabularies of related resources and their relations. RDFS and RDFS allow resources to be described in detailed descriptions and classified with a hierarchical structure [30]. Below is an RDFS-extended version of the previous example.

```
@prefix ad: <http://smart-api.io/ontology/ad#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ad:Tray1 rdf:type ad:Tray .
ad:Tray2 rdf:type ad:Tray .
ad:SmallerTray  rdfs:subClassOf ad:Tray ;
rdfs:label "SmallerTray"@en ;
rdfs:comment "A type of Tray"@en .
```

Other expressive technologies that are built upon RDFS such as OWL and SKOS focus on representing vocabulary structure and allow reasoning in the Semantic web [30]. OWL offers the ability to express the nature of data properties, such as symmetricity, and transitivity. Machines can uses those characteristics of data properties to infer more statements from an ontology.

```
@prefix ad: <http://smart-api.io/ontology/ad#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>

ad:Tray1 rdf:type ad:Tray .
ad:Tray2 rdf:type ad:Tray .
ad:SmallerTray  rdfs:subClassOf ad:Tray ;
rdfs:label "SmallerTray"@en ;
rdfs:comment "A type of Tray"@en .
ad:equalTo rdf:type owl:SymmetricProperty.
ad:Tray1 ad:equalTo ad:Tray2.
```

From the example above, since ad:equalTo is an owl:SymmetricProperty, machine can induce a new statement ad:Tray2 ad:equalTo ad:Tray1 from the ontology. OWL has many sublanguages and specifications, all of which focus on different aspects of semantic reasoning.

To query and manipulate RDF data, a query language is required. For that purpose, W3C standard in 2008 recommended SPARQL, a recent addition to the Semantic Web stack of languages [30]. SPARQL allows consumers to run queries on RDF datasets in a similar fashion as SQL for traditional relational databases. Below is an example of a simple SPARQL query.

```
PREFIX ad: <http://smart-api.io/ontology/ad#>
SELECT ?tray
WHERE {
?tray rdf:type c:Tray
}
```

In order to to create a semantic data model system, that would fit the needs of particular IoT applications while taking into account the core value proposition of the overall IoT concept, I introduce Smart API, and a linked data-based, object-centric, semantics-enabled, transaction-capable framework for building APIs for IoT solutions. I discuss Smart API in details in the next section.

2.3 Smart API

Smart API is a semantics-based data model framework for IoT applications [36]. IoT applications apply to a variety of smart devices, ranging from tiny and power limited microcontroller-driven sensors, to embedded Linux gateways and full-scale cloud platforms; each of which has unique data models and formats. The application area is broad and the uses and devices diverse significantly, therefore the industry has inherently suffered from lacking a consensus of data standards and compatibility.

Semantics focuses on bringing meaning to data, or in other words, focuses on building open data. A semantics data set not only can provide the data itself, but also what the data stands for. As modern IoT systems are striving for more advanced data processing, the possibility of automatic processing of the vast amounts of data has become a crucial requirement. Therefore, Asema has included semantic engineering into the IoT data processing to develop Smart API, an semantics-based framework for transferring and storing linked data. The core features of Smart API are:

- **Object centric**: The primary intention of Smart API is to carry remote objects between different systems [36]. Objects in SmartAPI represent both the physical devices and the programming abstractions of those devices. Smart API transparently processes data structures, the data sending and receiving over the network.
- **Semantics enabled**: Smart API is built on the foundations of the semantic web [36]. Common vocabularies of the semantics in Smart API removes data ambiguity and allows automatic instance unit conversion as data is transferred.
- **Transaction capable**: The SmartAPI transaction supports monetizing the application operations and data. The SmartAPI transaction supports cryptographically encrypted data transfer for confidentiality and non-repudiation [36]. SmartAPI transaction service works as an invoicing ledger.
- **Secure**: Smart API is fully crypto-protected [36]. In addition to HTTPS, Smart API provides encrypted and secure signing mechanism for all messages. Moreover, Smart API offers OAuth2 authentication protocol built-in support for better software engineering quality of life.
- **Linked data enabled**: In Smart API, everything is inter-connected. Concepts can be created and linked between objects, messages, and in any other configurations [36].

2.3.1 Smart API Design Principles

The design of Smart API attempts to achieve a holistic view of the data that fits the needs of particular IoT applications while taking into account the core value proposition of the overall IoT concept. It needs to serve the multiple needs of various forms while being semantically processed and mapped into actual software. Smart API borrows a lot from object-oriented programming, where everything is an object. An object in Smart API can have:

- **Properties**: define what the object is
- **Abilities**: what the object can do, for example, what functionalities a smart device has

Creating an IoT object in Smart API powered software to control and measure sensor devices is straightforward. These are translated into software terms as instance variables and methods. Measurement and manipulation

can be achieved by simple get-set methods. Sensors and actuators can also be identified easily without explicitly modelling. If a property is readable, it is a sensor. On the other hand, if a property can be written, it is an actuator. A read-write represents an actuator with sensoring ability. Property in Smart API consists of three core variables: Quantity, Unit, and Datatype. This design preserves the principles of Smart API design:

- Everything is modelled as an object
- Each measurable property of that object has quantity, unit, and datatype
- Sensoring means reading that property and controlling means writing that property

These principles cover the basic functionality of all sensoring and controlling applications in one uniform fashion. Besides, for data management, Smart API adapts the CRUD-N concept for API convention. CRUD defines the four essential functions of persistent storage: Create, Read, Update, and Delete. "N" or Notify, is an addition from the autonomous word, which is an operation for signalling when data changes occur. Moreover, Smart API defines the time span associated with every property to embody the time aspect of data.

2.3.2 Smart API Core Objects

In Smart API, all concepts are classified into a hierarchical tree, whose root node is Object. Core classes that inherit Object are divided into three branches:

Entities represent tangible things operated on IoT networks, such as devices, services, people. Entities are the targets of processing, whose properties contain the data to be processed. Entities are further broken down into two subclasses: AbstractEntities and PhysicalEntities. PhysicalEntities have physical properties such as dimension or weight, while AbstractEntities do not. Both PhysicalObject and AbstractEntities can be further subclassed, but such explicit subclassing has an insignificant impact on data processing. There are several subclasses of PhysicalObject and AbstractEntities built-in in Smart API, such as Person, Vehicle or Service.

Evaluations are the intangible, technical things in the network that act as data carriers, including requests, inputs, and outputs. Evaluations are the definers of processing. Their methods contain instructions on how to process

data. Evaluations split into five subclasses: Activities, Abilities, Inputs, Outputs, Messages, Provenances. Most of the network processing takes place by creating an Evaluation for the target party to process and attaching data to it in the form of Entities. The most typical procedure involves creating a Request that identifies the target party, linking an Activity that determines what the target should do, and packing the processing data into the Activity as a set of Entities

Property containers include all data models that give a standardised format to the properties of Entities. These classes ensure that the data is understandable with the help of structures and vocabulary. The standard container of such measurement is a ValueObject, which includes Quantity, Unit, and DataType properties.

Figure 2.3 shows a hierarchical architecture of Smart API Core Object Classes.

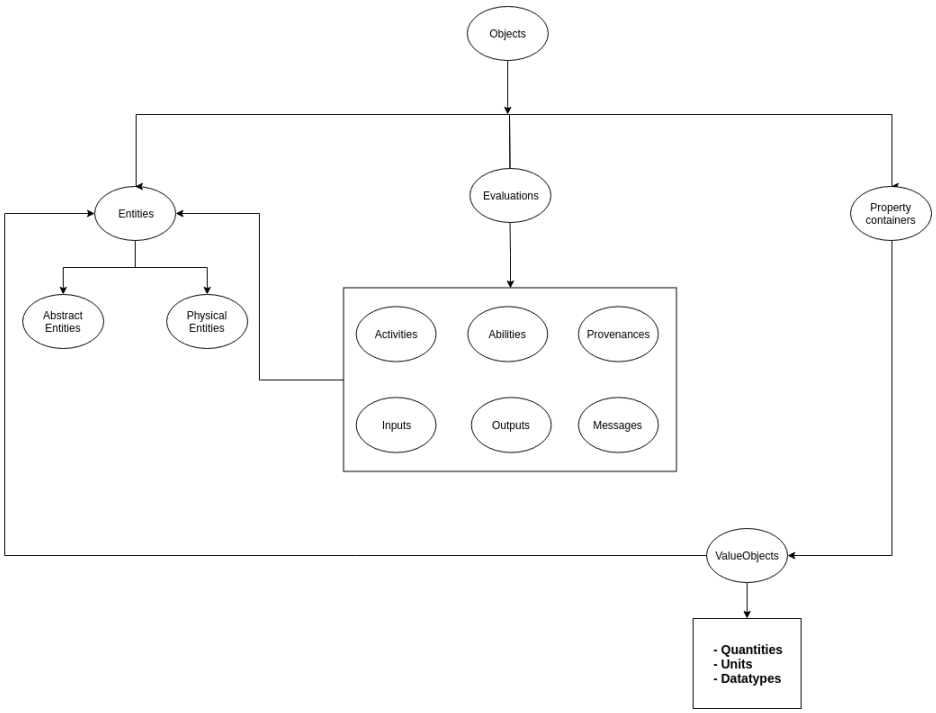

Figure 2.3: Smart API Core Classes

This hierarchical structure of classes enables comprehensive and self-explanatory data definitions, eliminates data ambiguity and enhances data reliability and maintainability. The Smart API adds identifiers into the data, which instead of being parsed as JSON, is now JSON-LD (JSON for Linked

Data, a serialisation of RDF) format. One example of a payload for a Smart API request is described as below:

```
{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "smart": "http://smart-api.org/ontology/1.0/",
    "nasa": "http://data.nasa.gov/qudt/owl/unit#"
  },
  "smart:weightBefore": {
    "nasa:unit": "qudt:kilogram",
    "rdf:value": 10^^xsd::int
  },
  "smart:heightBefore: {
    "nasa:unit": "qudt:meter",
    "rdf:value": 5^^xsd::float
  }
}
```

Chapter 3

Methodology

In this chapter, I discuss the research methodology applied in this thesis. I first go through an overview of the research process. Then I discuss each process step to understand why and how I use this methodology in the thesis.

3.1 Overview

The research process applied in this thesis work is a design science search process proposed by Peffers et al. in 2008 [37]. The research process is broken down into six separate phases, where human involvement is needed.

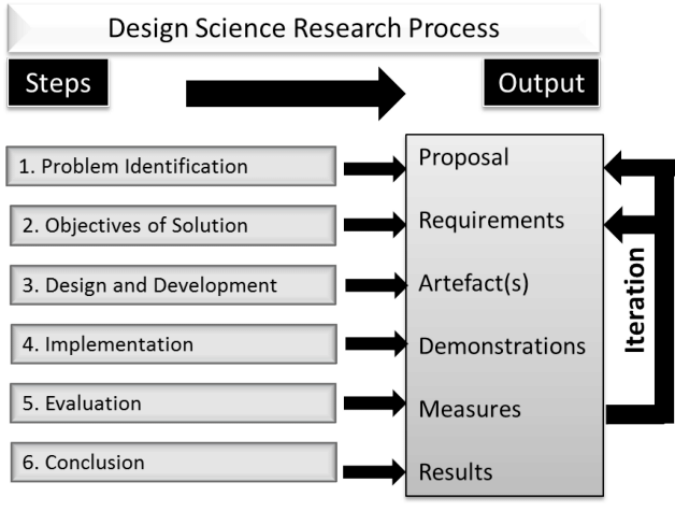

Figure 3.1: Design Science Research Process [38]

Figure 3.1 describes the design research process. The first phase is Problem Identification, where the problem is identified, and its practical relevance is ensured. The second phase is to define the scope and objectives of the research work. It is followed by the Design and Development phase, which focuses on designing and developing the goals and objectives that are set out in the second phase. The outcomes of the third phase are demonstrated in the Implementation phase, where the current designs and implementations are continuously iterated. Evaluation and Conclusion are the fifth and final phases, where a set of tests and evaluation criteria are run against the prototypes. Additional feedback from the team is also included. The execution order of the research process is not always sequential, but often back and forth between phases in an iterative fashion.

In the scope of this thesis, all six phases were applied, including implementation phases. In the following sections, I review each stage and the reasons behind all the design decisions.

3.2 Problem Identification

The problem is identified by the product owner and the team at Asema. During the beginning of the thesis, I held multiple weekly group meetings to help all team members understand the big picture as well as the nuances of the project. I tried to break down the research question into smaller subquestions, so that we could set out milestones for the thesis. After making sure everyone is on the same page, we agreed on the primary use case scenario to use as the foundation for the thesis work. All objectives of the problem were clearly defined in the use case scenario, which represents the definition of done of the thesis.

The target scenario includes a partner company who wants to share data with our system. There are two sources of data, a network of new sensors to be installed and an existing cloud-based ERP. Both sources of data need to be designed, defined and registered into the system. The ERP will be connected using the Smart API library. The data will contain two new data items that are not currently in the Smart API ontology, including one single-dimensional item, and one multi-dimensional item. The current version of the data designer is already able to generate single-dimensional data items, so building the multi-dimensional data item generation functionality is the main objective of thesis work. Moreover, the data designed must be able to offer a robust, intuitive interface to create both these items in a web browser environment and an ability to generate sample code that implements these designs.

Besides group meetings, we also have multiple workshops to gather input from all members working on the project. At the beginning of the workshops, each member presented the current state of their work, what was in their way, and what they were going to do next. Then we discussed the next steps to move forward in the project. We went through the requirements of and re-prioritised them based on the changes in the project.

3.3 Design and Development

Based on the requirements planned out in Problem Identification phase, I designed and iterated through two prototypes of the tool. Prototypes are an excellent way to test design decisions quickly and are the perfect match for development processes with iterations. Both prototypes were carefully evaluated with both expert opinions and user observation method. Expert evaluation is cheap and fast; however it relies on the expert who conducts the assessment. On the other hand, user observation takes more time, but it provides more accurate and quantitative insights on user interactions with the tool. I also checked if all requirement checklist items were fulfilled.

In the first prototype, I approached a form-based version, which was mainly for demonstrating the major functionalities of the final product. This prototype consisted of many forms with complex input fields where users can use to create and edit ontology. Though it proved to be very straightforward and fast-to-use for expert users, non-expert users were reported to have usability difficulties while interacting with the tool. It was mainly because expert users are familiar with the form-based approach, as it is used in many other existing tools.

However, as I aim to offer a fully-fledge solution for any users, regardless of their level of knowledge of semantic engineering; in the second prototype I come up with a graph-based solution, where the ontology was rendered in an interactive graph. Users can make changes to the ontology directly on the graph, and the changes will then reflect on the graph immediately. The prototype was evaluated with the same set of testing cases, and it showed that users at any level could interact with the tool fast and effortlessly. The majority of users reported that graph-based version has a much more smooth learning curve, and they were able to get onboard in significantly less time.

Based on the evaluations of both prototypes, I came to the conclusion to go with the graph-based approach. The design and implementation of this installation are discussed in the next chapter.

Chapter 4

Design and Implementation

This section outlines the design of the system architecture and the implementation process of the proposed solution. I first review the Asema IoT system architecture and the Smart API framework to give an overview of the thesis work. I then discuss the design process of the ontology presentation in a graphical user interface and its operation in the Smart API data network. Finally, I analyse the use cases, functionalities, underlying technologies, and the implementation of the ontology editing tool.

4.1 System Architecture

The Asema IoT Central is an adaptive system integration software for Internet of Things systems. It differs from the traditional IoT solutions, which are heavily reliant on middleware architecture, by having a lightweight, extendable modular application design. The modular architecture allows Asema IoT Central to scale from simple-machine application to fully-fledged service-oriented cloud systems effortlessly. Moreover, the lightweight approach enables hardware and power efficiency, painless maintenance, and native support for edge computing. This architecture also allows out-of-the-box integration for sensor monitoring, remote control dashboards, product pricing, information displays, and many more services, whose sources are included in the application. Users can modify, expand, or extend the services for re-branding or full business logic customisation.

Being designed as a building block for custom application integration, the Asema IoT Central can flexibly connect to other network entities, forming a grid to perform distributed tasks. This design not only enhances the overall system scalability, but it also helps answer the questions of where data is located and under whose possession. In the Asema IoT system, data access

can be set by location and shared with either temporary or permanent access rights. Figure 4.1 demonstrates the high-level architecture of the Asema IoT solution.

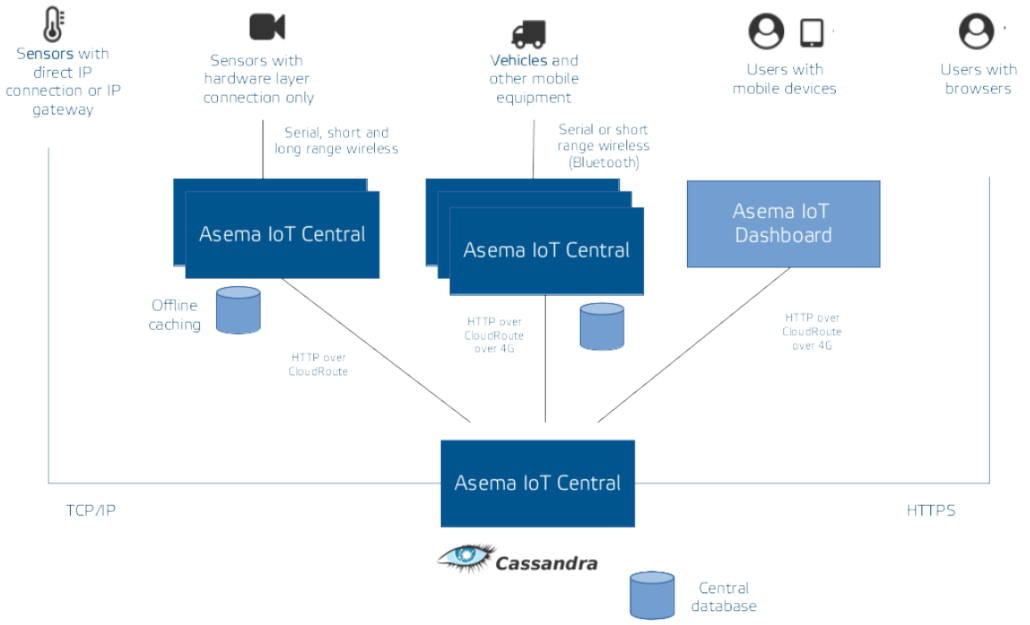

Figure 4.1: Asema IoT Solution High-level Architecture [39]

Although the architecture of Asema IoT Central is very flexible and works natively in many use cases, there is no one-size-fits-all solution. Therefore, Asema IoT software supports three types of installation, namely Asema IoT Central, Asema IoT Edge, and Asema IoT Dashboard. All of three installations include the same object-centric, event-driven core architecture so that they can integrate effortlessly in large scale systems:

- **Asema IoT Central** is a fully-fledged, cloud-based IoT server and client implementation for data collection, data sharing and system management. It includes business logic, local and remote user interfaces, analytics interfaces, and client interfaces. By leveraging the latest database and networking technologies, Asema IoT Central fully supports both traditional relational databases and modern NoSQL databases. It can operate seamlessly in either standard network protocols (HTTP, MQTT, CoAP) or local wireless or wired connection (WiFi, Bluetooth). Moreover, Asema IoT Central features Screen-let Store, which is an app store for effortless online IoT applications distribution. This installa-

tion supports the major platforms and operating systems, including Windows, Linux and Mac OS.

- **Asema IoT Edge** is a lite version of Asema IoT Central for edge computing. It also contains the core business rule engine, database support and integration components, but only supports basic user interfaces and no data analytics tools. This is a compact version designed to run on memory-limited devices and can be installed into embedded environments. It is compatible with embedded Linux distributions and can be cross-compiled to ARM architecture.

- **Asema IoT Dashboard** is the implementation version of IoT Central for mobile devices. It is a user interface component, which does not have any database, business logic or server supports. However, Asema IoT Dashboard offers a sophisticated, fully customizable graphical user interface for IoT applications. This installation runs on Android and iOS.

The flexibility of software installations is crucial for mass adaptation. The application is available in ready-made builds for all major operating systems. All have been tested in operation on a wide range of virtualization technologies, including VMWare, VirtualBox, Microsoft Azure and Amazon EC2.

The JSON RPC API is a format used in many modern network services. A standard JSON RPC call consists of an ID, a method name and parameters. The method in the payload represents the actions of the API calls, which can be searching, fetching or any controlling actions. An example JSON call to fetch all objects in the system is described below.

```
{
  "id": 1,
  "params": Null,
  "method": "fetch_objects"
}
```

With the context provided by Smart API, the payload clearly defines what the data is, which is highly beneficial for software developers in their development process. Moreover, to help cutting the integration overhead, Asema offers a free, open source Smart API SDK in all major programming languages. One core component of the SDK is the data designer tool, which allows users to design their data models in an intuitive graphical user interface. The design and implementation of the tool are discussed in the next sections.

4.2 Design of the Data Designer

The first part of this section describes the architectural design and the specifications of the solution, where I thoroughly go through how system components communicate with each other and how data flows between them. The second part focuses on the use of case analyses and the design plan.

4.2.1 Requirements and specifications

In Internet of Thing systems, where many different parties are continually collecting and storing raw IoT data, it is not easy to justify whether the collected information is beneficial or not. In many cases, IoT applications collect raw data under an assumption that the data can potentially become useful in the future. In today world, the costs for collecting and storing data are becoming more trivial, the risks of missing out on some valuable data outweigh the costs. Therefore, many companies chose to pile up their data instead and sometimes leave it in dust for years, because for IoT data to be useful, it has connected with other data. However, data structures in IoT are very complex, and it is very easy to lose track of what includes in a data collection. The database was probably designed and maintained by a data engineer who left the company or stopped working on the project long ago, and manipulation of data is left hanging and replies solely on documentation.

On the other hand, software engineers, who are building applications usually neither have the skill nor the access to the actual format of data. This issue causes overhead in data integration processes. The solution is primarily designed to address this problem between systems. Smart API is a semantic-based API framework, that allows the engineers of an organization to use APIs with minimal support from the engineers at other organisations, and provides standardised APIs when their systems are interconnected. To do that, both the requests to and responses from the APIs must be self-explanatory.

Based on the understanding of the system requirements, I found that to provide an efficient solution; the designed architecture has to acknowledge the fact that different actors have different data domains. The platform architecture is designed to ensure the availability and consistency of data by introducing microservice architecture with semantic interoperability of systems.

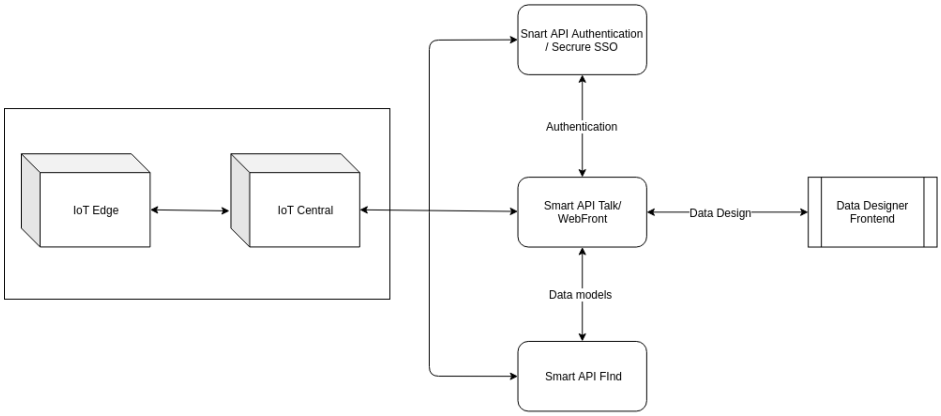

Figure 4.2: Platform Architecture

Figure 4.2 demonstrates the platform architecture and communications among system components. The structure and functionalities of each component below are described below.

WebFront is the central service. It is a standalone application, which acts as a proxy, connecting different services and maintaining the communications with these components via several interfaces. WebFront also provides a set of backbone APIs, serving multiple front-ends, including the Data Designer. When users make interactions with Data Designer or other front-ends, the request is submitted to WebFront, from which it is forwarded to the corresponding services that are responsible for handling those requests. The services then process the request and respond to WebFront, which then send suitable feedback to users.

Find Service or **Smart API Find** acts as a navigator in the Asema global semantic map. It stores and maintains information about devices connected to the systems, where they are located and under whose controls. Control systems can find data providers, and service providers can discover control systems via Find Service. Moreover, It represents systems when a third party service is required to relay data such as state information.

Talk Service or **Smart API Talk** maintains the Smart API data model.

The service provides developers with online tools to further develop and release data versions, testing facilities to test data validity, and an SDK to help integrate data model straight into their software development process.

SSO Secure Service or **Smart API Secure** offers a uniform authentication method to the whole system. It issues identities to participants in the form of the authentication process so that users can use one username and password to identify themselves in multiple applications. Furthermore, Secure Service operates as a policy distribution central hub, allowing known and unknown parties to negotiate modifications in their security policies.

This architecture is designed to ensure availability and stability and does not have any single point of failure. The system scales up or scales down can be achieved effortlessly by adding or removing nodes of any components. Moreover, the data model is stored in a single source of truth to establish data consistency and discoverability.

I review the use cases of the data designing process in the next section.

4.2.2 Use case analyses

The main objective of the data designer is to allow users to explore and create new classes of objects, which serve as global search terms in the Smart API Find directory. In addition, other users can also find those newly added concepts from the Smart API Talk server and use them to create their ontology or to build compatible applications. Two groups of users are projected to be the primary end users of this tool in the scope of this thesis. The first group is technical users who are more familiar with ontology engineering and semantic web, but necessarily technical experts or having intensive domain knowledge. The other group covers non-expert users, who have little to none experience with semantic web and ontologies. Despite the gaps between technical understanding and capabilities of two groups, this thesis aims to offer both of the groups two main goals.

The first goal of the Data Designer is to provide helps in the software development process. It allows users to create, edit, and extend the data model in an intuitive graphical user interface. Besides, the Data Designer can generate ready-to-use source code stubs that can be copy-pasted into the applications to shorten development time and reduce repetitive works. The tool also presents clear documentation and explanation for the data model, so that software developers can use the generated code stubs out of the box in their software development, cutting a sizable amount of time otherwise will

be spent on studying the domain knowledge and understanding the library. This approach helps ease the learning curve of industry knowledge as well as software practices and provides standardised coding references. Developers also can share their data models easily by exchanging generated code stubs.

Secondly, and most importantly, the Smart API Data Designer makes it possible to share data models between organisations. One major problem in the IoT data interoperability is the terminology and unit disparities among organisations, where different terms and units are used for the same thing. The rate of movement can be identified by "Speed" and measure by "Kilometer per hour" at one organisation while being "Velocity" and "Meter per second" at another organisation. This issue causes severe disorganisation, turbulence, and complications during system integration processes. Data Designer tackles this issue by enabling a universal ontology sharing model. Once a terminology is created, it is available on a common vocabulary, which is shared among all parties, while all units come with a built-in conversion module. The common vocabulary is a systematic hierarchical graph of ontologies. Developers can use smart searches that suggest both exact word searching and synonym searching to explore the common vocabulary for the appropriate terms and units that fit their particular system. They then can choose to reuse the existing terminologies, or copy/extend them for further modification. Furthermore, all parties in the Smart API network are assigned to and identified by unique namespaces to avoid data conflicts.

We analyse the use cases based on the understanding of these goals. Use cases can be classified into three groups: Data Model Design, Data Sharing (Data Exchange), and Authentication. Figure 4.3 displays the use case diagram of the Data Designer tool.

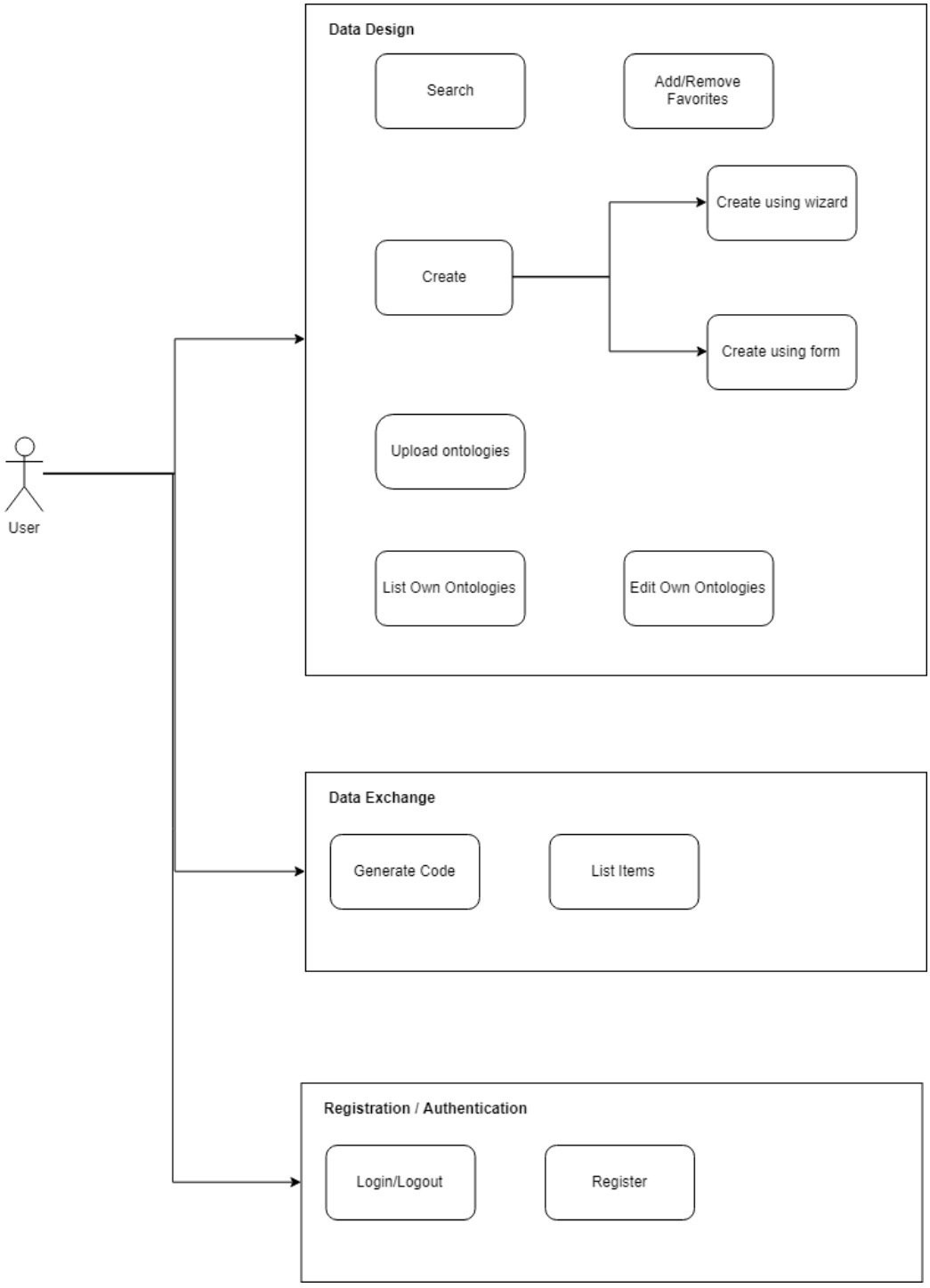

Figure 4.3: Use Case Diagram

Data Model Design group consists of use cases that are related to the data creation and data management processes: **create concept**, **upload ontology** from local computer, **list concepts**, and **edit concept**.

Create Concept is the core functionality of the tool that allows users to create and submit data concepts to the common ontology. Table 4.1 shows ten types of concepts that developers can create. As we discuss in the system requirements and specifications section, there are two distinct user groups that are using the tool, each of which has a different level of expertise in semantic and ontology engineering. Therefore this function is designed to be flexible yet standardised for both user groups. After interviewing users from both groups, we recognise that while a form-based approach is more suitable for more-expert users, a graph-based user interface can help non-expert users to bridge the ontology knowledge gap and be able to focus on creating data models. The tool supports both approaches, from which users can choose either one that is more suitable for them. User can create the following types of concepts:

- **Creating a Quantity**: Define a quantity, which is the unit of measure. It represents a quantitative property for the object. For example, a quantity can be the length, speed or temperature
- **Creating a Unit**: Define a unit, which is used to measure quantity. For example, "Length" quantity can be measured by "meters" or "kilometres" units
- **Creating a Device Type**: Define a type for physical devices
- **Creating a Service Type**: Define the type of connected service
- **Creating a Data Type Property**: Define data type used for value objects. For example : "string", "integer", "number", etc.
- **Creating an Object Property**: Define a relationship between concepts. For example: "hasWeight" can be used to link an object to its weight
- **Creating a Unit Category**: Define a category of units. For example: "Kilometer" and "Meter" can belong to the "LengthUnit" unit category
- **Creating a Quantity Category**: Define a category of quantities.
- **Creating a General Class**: Define a class of objects. For example "Car" can be a class

- **Creating a Class Member**: Define a member or an instance of a class. For example: "MyCar" can be an instance of class "Car"

Upload Ontology allows users to upload ontology files from there local computer to generate a full ontology defined in the uploaded files. Supported formats include Turtle, JSON-LD, and N-Triples. The uploaded ontology definition files then are tested and validated in the Testing Service, controlled by WebFront server. If the imported data is valid and not existent in the common ontology, it is added to the central graph system under the owner's namespace. If the namespace does not exist, a new namespace is created and assigned to the owner. If there is a conflict, users can choose to either import the ontology to a new namespace or cancel the upload.

List own concepts and **Edit concept** are designed for managing created concepts. After logging in, a user can see a list of their established concepts in a connected graph, where each node represents a concept. Users then can select a concept to edit, and the graph is automatically updated to match the changes. Users only can see and edit concepts under the namespaces that they have administrator rights.

While Data Model Design use case group covers data model designing and data model management processes, users also need to be able to share their created data models. There are two use cases for this group, which are **Search** and **Generate Code**.

Search is critical for exploring and sharing data. The quality and consistency of data models in common vocabulary rely heavily on the quality of search results. If developers are not able to search for the concept they are looking for, they will create a new data model for that concept. The system, therefore, is prone to data redundancy and duplication because multiple models are created to represent the same entity. Moreover, the search function is required to be able to proactively suggest developers existing terms that might be suitable for their needs during every step in the data model creation process. This search also applies to unit search and datatype search. We design a smart, self-autocomplete search module that looks up for concepts that match the exact input keyword, as well as concepts that are related to the keyword and its synonyms. The implementation of this module is further discussed in the Implementation section of this chapter.

Generate Code is for generating code stubs. After defining an ontology,

users can build reusable code stubs in selected programming languages. The code stubs include all concepts in the ontology and are ready-to-use with simple copy-paste. The supported languages include C++, Python, Java, and PHP.

In the next section, I discuss the implementation of this design.

4.3 Implementation

In the scope of this thesis, the implementation consists of a single page app user interface and a data model management back-end server. The application of a graphical user interface covers all use cases analysed in the previous section, and the back-end server handles the underlying logic behind user requests. This section first gives an overview of the data designer tool and its components. Then, I discuss front-end and back-end implementations in detail to justify the software architecture choices.

4.3.1 Overview

The main goal of the graphical user interface is to provide a responsive, modern workspace that is intuitive and simple to use so that both technical and non-technical users can interact with effortlessly. Figure 4.4 describes the main screen of the graphical user interface.

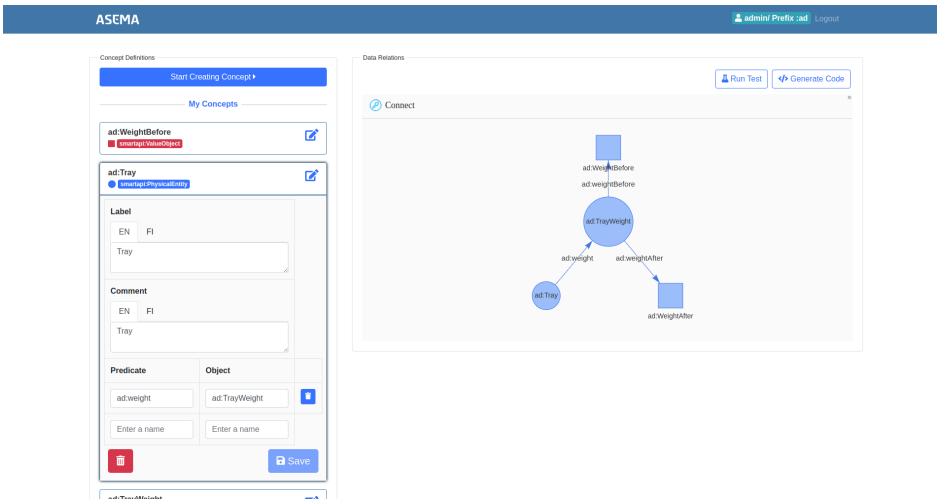

Figure 4.4: Data Desginer Graphical User Interface

The main components of Data Designer GUI are: create concept form wizard, ontology graph, quick edit tables, and code generator widget; each of which is responsible for a set of core functionalities. Users are required to login to use any functions in the tool. After logging in, username and prefix are visible on the top bar navigation bar, and users can start interacting with the tool.

Figure 4.5: Top Bar

The fundamental element that is utilised throughout all components in the tool is the smart search input. It is an input element that provides a list of suggestions when users look up for a keyword so that users know what is already in the system that can be reused, and what is not. Upon the user's typing, the smart search input automatically talks with the ontology server, retrieves the suitable recommendations, and displays them to the user. Moreover, the smart search provides users with quick add-on actions based on the context of use. Figure 4.6 demonstrates a smart search input element.

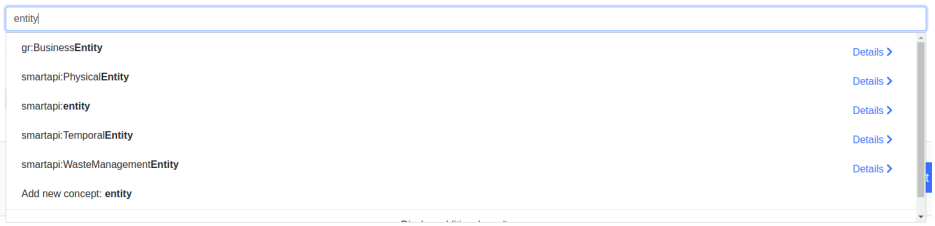

Figure 4.6: Smart Search Input

The component that is most beneficial from the smart search input is the concept form wizard.

4.3.1.1 Concept Form Wizard

From the homepage, users can enter the concept form wizard by clicking on the "Start Creating Concept" button. There are two separate steps in the form wizard. The first step is the initialisation step, where the user setup the concept name and its parent class.

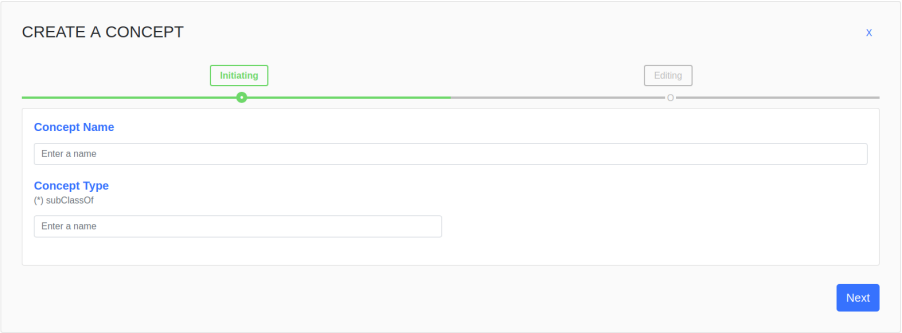

Figure 4.7: Create Concept Form Wizard

As we discuss in the requirement section, it is important to make sure that users do not create a new concept for an existing entity in the system, so in this step, when users enter the concept name, we provide a list of existing concepts that can be used instead. Users can review the details of those existing concepts by clicking on "Details" on each recommendation item. A popup with all details of the selected concept shows up as below:

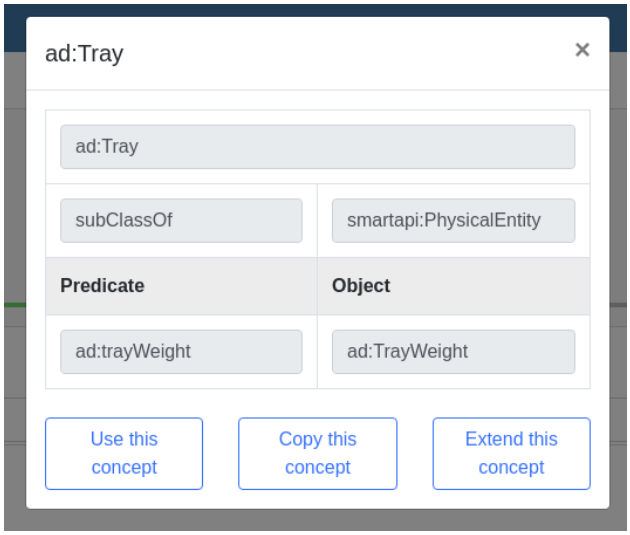

Figure 4.8: Concept Detail Popup

Users then can choose to use, extend, or inherit that concept. Copying concept creates a new concept that has all properties of the original concept, and users can add more statements without affecting the original one. On

the other hand, extending concept creates a subclass of the selected concept. If they cannot find any concepts that match their needs, users can continue to create a totally new concept. However, in order to keep all concepts under a hierarchical tree, users also need to select the parent class when creating a concept. After filling in all required fields in step one, users can start editing the concept in steps two of the wizard.

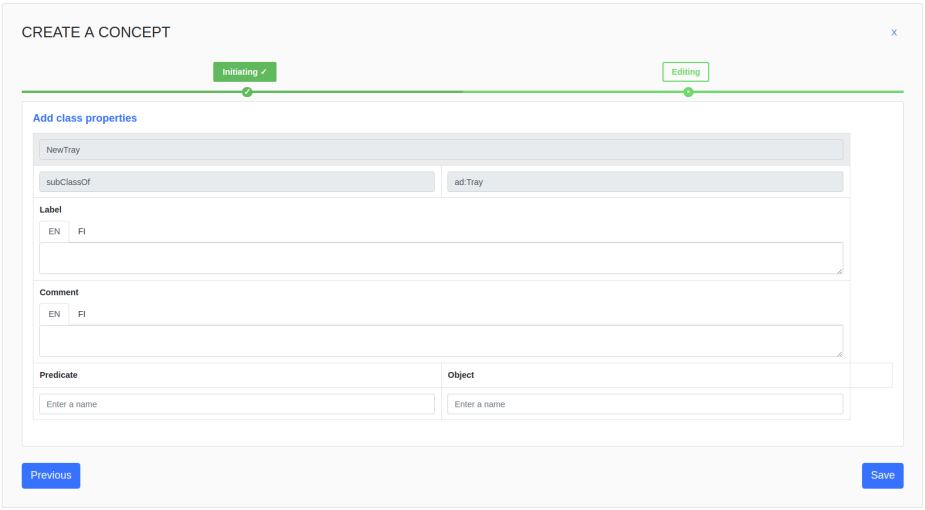

Figure 4.9: Concept Editing Form

At least one label and one comment in any language are required. Users also can add properties to the concept by giving statements with predicates and objects. After saving, users can choose to either continue creating a new concept or review their ontology namespace in the ontology graph on the homepage.

4.3.1.2 Ontology Graph

The ontology graph is the most critical component to provide users with an intuitive view of their ontology. It visualises all current user's concepts as well as their relationships in graph panel and allows the user to interact with their ontology directly. The nodes are the concepts, and the edges are the relationships between them. Different shapes represent different concept types. For example, Physical Entity concepts are shown as circle nodes, while square nodes stand for Object value concepts. When a new concept is created, it is also automatically added to the ontology graph.

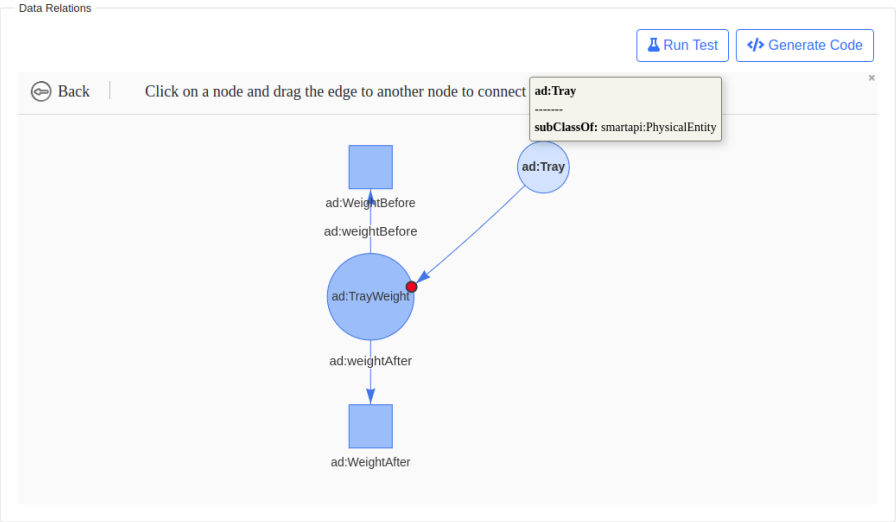

Figure 4.10: Ontology Graph

The ontology graph supports all basic graph functionalities, including drag and drop, zoom in, zoom out and hover to see details. Furthermore, users can create connections between their concepts directly on the graph. Users start by clicking the "Connect" button and then drag a connection arrow from the subject concept to the object concept. The ontology graph updates and takes care of the validations and interactions with the server in the background, to ensure that the user experiences is as smooth as possible without any blocking.

Besides, while the ontology graph aims to bring helps to non-expert users, the tool also offers form-based quick edit concept tables for users with more experience with ontology.

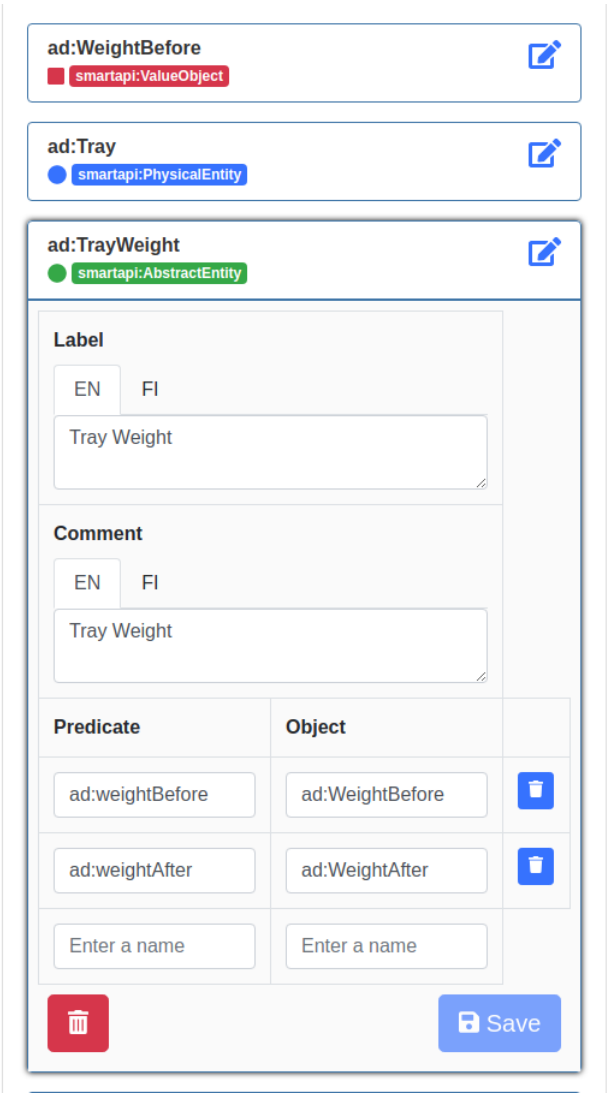

Figure 4.11: Quick Edit Table

Quick edit tables work in sync with the ontology graph. Whenever users make changes in one of the two components, both of them are updated.

4.3.1.3 Code Generator Widget

When users click on the "Generate Code" button, a modal window shows up, users then can choose which concept to generate code stub, to which programming language, and under which code type. The supported programming languages include C++, Python, Java, and PHP. Users can use

this code stub out-of-the-box in their software or share it with others without any modifications. Figure 4.12 shows the code generator widget window.

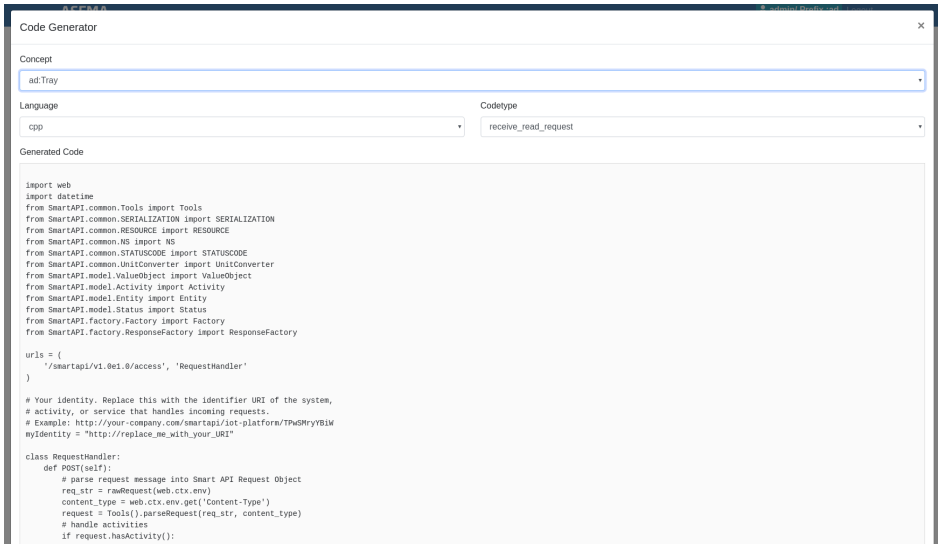

Figure 4.12: Code Generator Widget

Those components cover all use cases analysed in the Design section of this chapter. I discuss the implementation details and chosen technologies in the following sections.

4.3.2 Frontend Implementation

To offer a fluid user experience plays a vital role during the frontend designing and implementing processes. One main goal of the thesis is to build a modern looking, simple to use GUI that users at any technical level can interact with effortlessly. I choose web browsers as the working environment for the data designer because with the help of cutting-edge Javascript technologies, web applications are becoming faster, more convenient to use, easy to update, and device-independent. Figure 4.13 describes an overview of the frontend architecture.

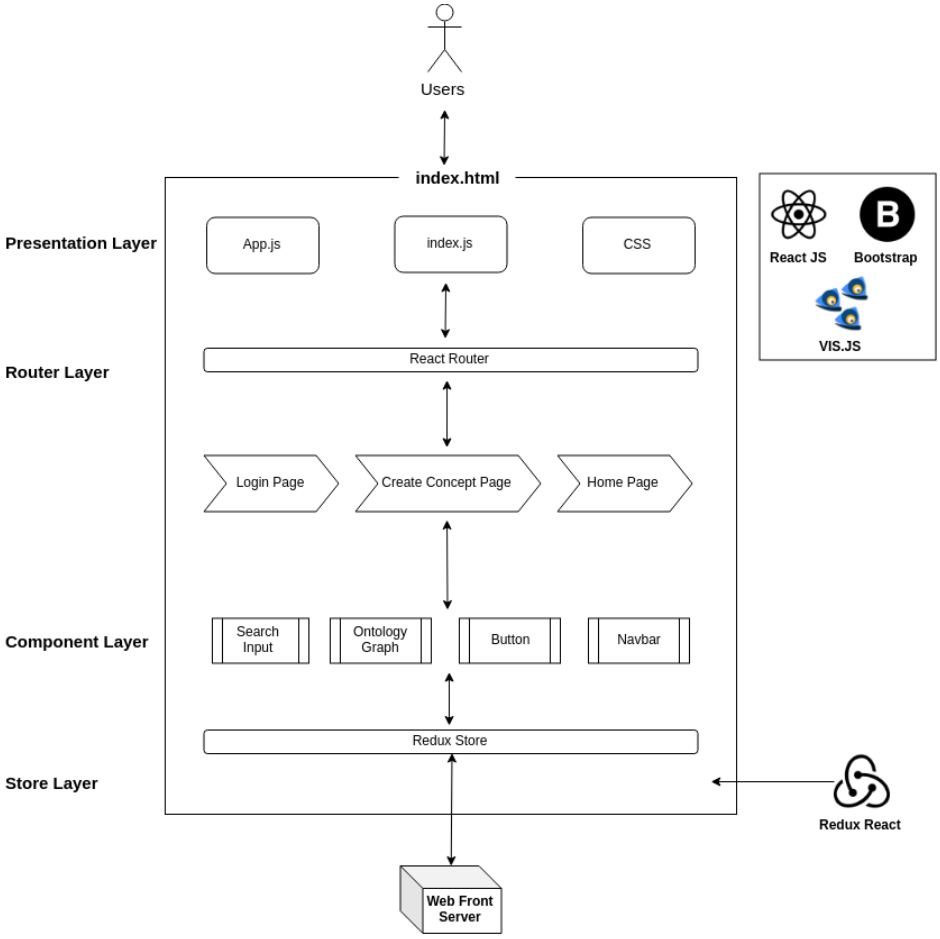

Figure 4.13: Frontend Architecture

In the web environment, response time or waiting time between request response circles could be problematic for IoT systems that process large data sets with complicated logic. Users cannot have good user experience if they have to wait several seconds after clicking a button when instantaneous feedbacks are critical for nowadays applications. To address this issue, I apply the single-page application design pattern in implementing the designer tool. A single-page application is an application that works under a web browser environment and does not require page reloading during use. Different from the traditional multi-page application, single-page application loads all web resources, including HTML, CSS, JS scripts only once throughout a web session, and dynamically loads data from servers, rewrites the current page rather than re-rendering entire new pages. This approach minimises the blocking of the user experience between page transitions, making the appli-

cation works more smoothly and comparable to desktop application performance. Moreover, with the help of modern web frameworks, the development of single-page applications is streamlined and simplified. I choose React, an open source Javascript frontend framework created and maintained by Facebook, as the framework for building the designer tool. A React application is divided into independent, reusable building blocks called Components. This design principle also fits our needs perfectly, as many components in the designer are used in different locations, such as the smart search input, concept table.

Besides, to offer users an interactive interface, I leverage AJAX and asynchronous programming techniques in processing HTTP requests. AJAX allows web applications can send and retrieve data from a server asynchronously in the background, without blocking the display and behaviour of the application, therefore enables a more fluid user experience. The tool handles all requests asynchronously and stores all data a common store using Redux React. However, it always returns acknowledgement feedback first so that users know that their requests are received and being processed. This approach, together with single page application architecture provides a robust and uninterrupted user experience. Moreover, to give the GUI a modern look, we build the web interface on Bootstrap, which is an open source CSS framework, that focuses on building responsive, mobile-first applications. I apply flat design principles to all components to create a consistent and unique user interface. For developing the ontology graph, I examine the vis.js library, which is a dynamic, browser-based visualisation library. It offers many features to manipulate and interact with large amounts of dynamic data.

I review the implementation details of the backend Web Front server in the next section.

4.3.3 Backend Implementation

On the backend side, I use Python to build the WebFront server and many other services. It has been the programming language of choice when it comes to web development for many years. Python offers:

- Fast and robust web development
- A vast amount of libraries and fast growing community
- Stable and powerful scripting capabilities that allow software engineers to develop more with less code.

- Supports asynchronous networking, which is critical to work with single-page frontend applications
- Open-source and mature programming language

Although Django is the most popular Python web framework, it also contains many components and features that are not beneficial for this thesis. Therefore, to avoid redundancy, I choose Tornado as the backbone framework for the WebFront server. Tornado is an open source web framework that is not as feature rich as Django, but smaller in size and might run faster. Moreover, Tornado supports non-blocking network I/O, allowing it to scale to thousands of open connections effortlessly. This feature proves to be extremely valuable for an IoT web server.

Another important task of the Web Front server is to process and maintain the common vocabulary. There are several Python RDF processing libraries, including 4RDF, RedlandRdf, Rx4RDF, and rdflib. While 4RDF is progressively and merged into rdflib, RedlandRdf and Rx4RDF are outdated and complicated to use, rdflib has emerged as the most reliable and powerful python tool to work with graph data. I use rdflib to maintain and manipulate the RDF data in the system, and store all ontology in files. The web server loads all those ontology files into a graph store during the initialisation. Figure 4.14 shows the architecture of the backend server.

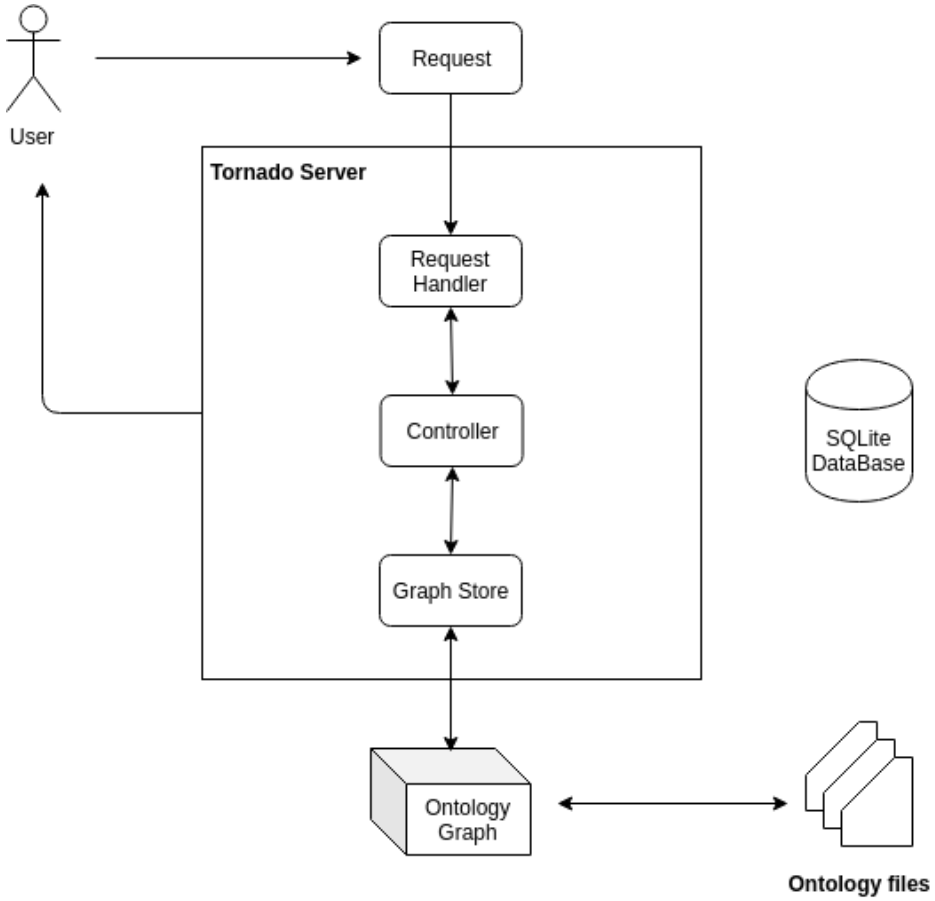

Figure 4.14: Backend Architecture

When an API request comes to the Tornado server, it first reaches the request handler module. Request handler module acts as a router, which matches API requests with responsible controllers. The matched controller then unpacks request payload, processes it by talking to the graph store. After processing the request, the server returns a response to the user in standard JSON format. The back-end server processes requests from the front-end asynchronously to make sure there is no interruption in the user interface.

To justify this approach, I benchmark both frontend and backend implementations under a practical use case scenario, which is discussed the evaluation of the system in the next chapter.

Chapter 5

Evaluation

In this chapter, I evaluate and analyse the proposed implementation discussed in Chapter 4. First, I briefly present the evaluation criteria and an example use case. Then, I consider the research questions presented in the Introduction chapter, and thoroughly analyse the solution to see how it complies to the criteria and what has been accomplished in the testing context. Finally, I benchmark our proposed solution with existing tools in the market.

5.1 Evaluation Criteria

As I discuss in Chapter 1, the primary objectives of the thesis work are:

- To improve the overall usability of the data designer tool: achieving this would answer the main research question, which is how to present information in an intuitive way that makes it easy to create, edit and share ontology-based data in IoT industry, to improve the interoperability and data compatibility among different parties.
- To implement multi-dimensional data design processes: this is the major improvement of the new version of the data designer tool. The current version of the tool does not support multi-dimensional data models, which limits users to only simple single-dimensional data models.

In order to verify the objectives, I planned out a use case for testing. The testing use case consists of a cloud-based ERP, which is connected to Find Server using the Smart API library. The data will contain new multi-dimensional data items that do not exist in Smart API ontology. The data designed must be able to support multi-dimensional data creation and editing

in an easy to use web dashboard. Moreover, when the data design is done with the dashboard, the dashboard needs to be able to automatically generate Smart API SDK compatible sample code for the design.

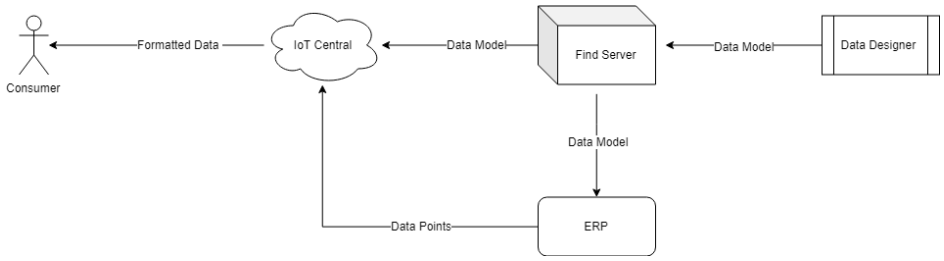

Figure 5.1: Evaluation Scenario

I evaluate the proposed solution based on those requirements in the next section.

5.2 Evaluation

Overall, I have accomplished the two main objectives set out at the beginning of the thesis work. The usability of the tool was enhanced significantly. I have revamped the whole user interface, giving it a new modern, intuitive look. The most significant improvement in the GUI was that ontologies are visualised in an interactive graph so that users can have a graphical overview of their data models and their relations. Figure 5.2 demonstrates the changes in the GUI.

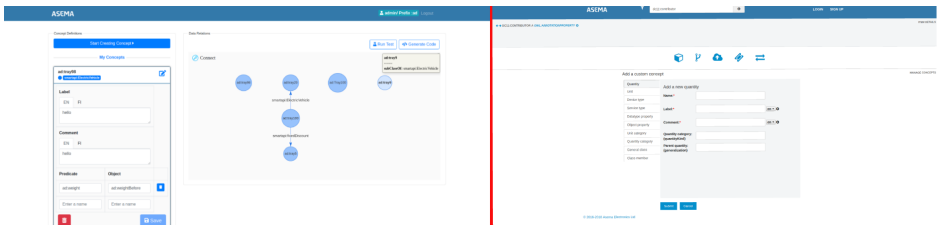

Figure 5.2: Changes on the graphical user interface

One of the components that are enhanced the most is the create concept form. In the previous version of the tool, adding concept requires many steps in many different screens, and users have no overview or general understanding of the progress. The user interface is confusing and not self-explanatory.

There are no prominent indicators to tell users what they can do on the screen, or where they can start the concept creating process. During a usability test, some users report that they do not know where to begin or how to use the tool. Moreover, when users create a new concept, there are no suggestions for existing concepts that can be reused, therefore limiting the reusability and hierarchical structure of data models.

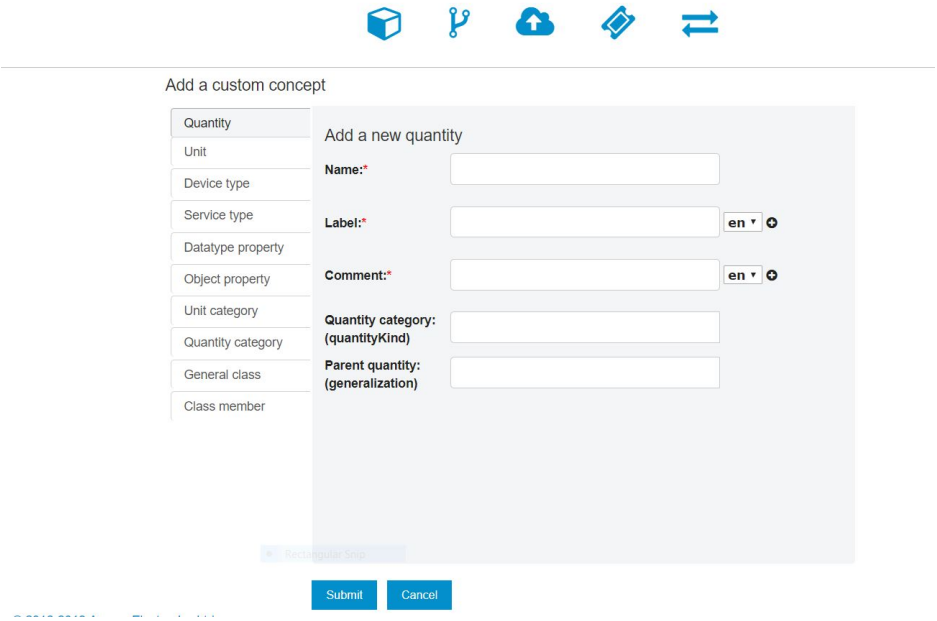

Figure 5.3: Old create concept form

In the proposed solution, I focus on improving the defects found in the usability test of the previous version of the tool. The whole process is presented with guidelines and pages are broken up into clearly defined areas. The new tool provides a clear visual hierarchy on each page with prominence headers and colour-coded step indicators. Users now can see clearly where to start, where they are currently at, and what would be the results. In addition, all inputs now are empowered with autocomplete and suggestion functionality so that users can take advantage of existing concept base. Thanks to the changes in the UI, the same user group said that this is much more intuitive and they was able to create concepts without any human instructions.

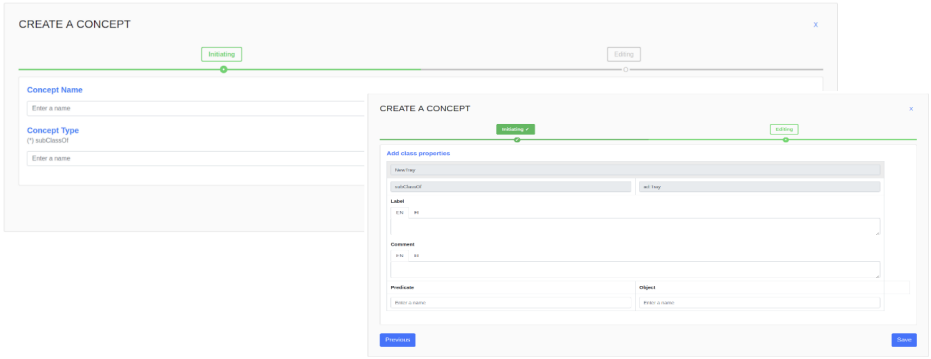

Figure 5.4: New create concept form

In addition, in the older version of the tool, to interact with data models, users have to access different pages with a staggering number of forms and inputs. In the proposed solution, users can do everything in quick edit tables with complete helping accessibility, such as auto-complete inputs, drag and drop actions. The number of inputs and forms is reduced noticeably. Besides, I introduce many new data model manipulation functionalities in the proposed solution. Most importantly, ontology now can be visualised in the interactive ontology graph, providing users a graphical overview of data models and their relations, and also allowing intuitive interactions.

Functionality	Old Version	Proposed Solution
Create data model	Requires 3 different views at 3 different places	Everything can be done on only 1 view
Create data relation	Unavailable	Can be done while creating data model or on the graph
Edit data model	Difficult to find where to start editing	Can be done on the graph or with quick edit data table
Delete Data model	Unavailable	Can be done on the graph or with quick edit data table
Data Visualisation	Unavailable	Data is visualised in an interactive graph
Multidimensional data model	Unavailable	Fully supported

Table 5.1: Functionality comparison

Moreover, not only the user interface has been fully renovated, but the performance of the tool has been improved remarkably. The loading time was reduced by 31.4 percent, from 2217ms for the first time loading to 1520ms. The average response time was cut by 43 percent, from 144 ms to 82 ms.

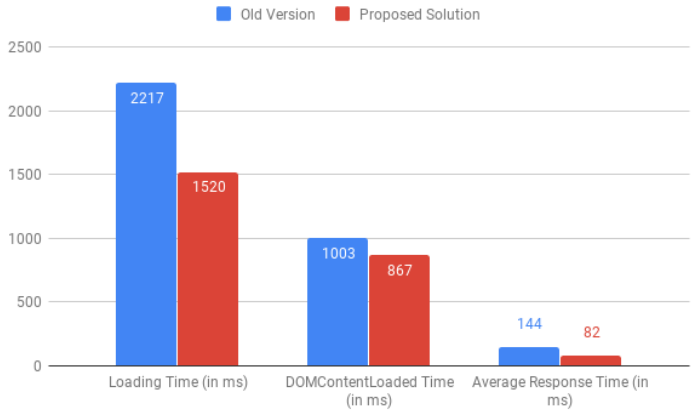

Figure 5.5: Loading Time comparison

Moreover, since the new tool is a single page application, users do not have to reload the whole page when submitting a request, but only needed data. The data transferred was also decreased drastically, over 60 percent from 1.6Mb to 667Kb.

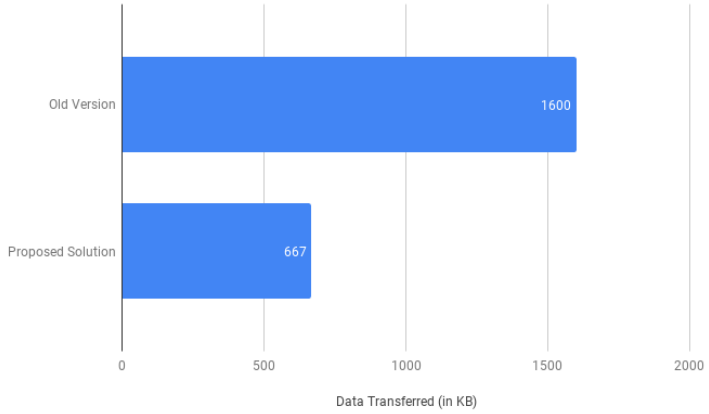

Figure 5.6: Data Transferred comparison

The other primary task of the thesis work, which is to implement multi-dimensional data model support was also finished. Users can create a multi-dimensional data model effortlessly. A data model now can have both property relation with value objects and data relations with other resources. Data relations can be created on the fly on the graph by dragging a new edge from

the subject resource to the object resource. The edge represents the predicate of the statement.

Under the testing scenario, Data Designer tool was able to register data models to Find Server, which is visible for ERP and IoT Central. Users can generate code stubs from those data models on the graphical user interface easily. The proposed solution successfully demonstrated that it can work in sync with other services in the system.

5.3 Existing ontology editors

Over the years, Semantic Web technology has been beneficial from an array of data management tools, glossaries, ontology and vocabulary building platforms. In a semantic system that includes multiple parties with complex data flows, the job of creating, editing and sharing ontology models is burdensome. A number of tools have been introduced to tackle this problem. We discuss the existing tools specialised in ontology editing below.

Protégé [40] is an open-source platform, including a suite of tools for domain model construction and knowledge based engineering with ontologies. It has been developed and maintained by Stanford University. Protégé supports the OWL 2 Web Ontology Language comprehensively and offers a feature-rich environment for ontology editing. Protégé features include create, upload, modify, and share ontologies for collaborative viewing and editing in a fully customizable user interface.

NeOn-Toolkit [41] is another open-source, multi-platform ontology editor, which offers vocabularies creating and publishing compatible with Linked Data principles.

TopBraid Composer from TopQuadrant [42] is the most reputable candidate on the commercial product side. It is a W3C standard-compliant platform for building Semantic Web ontologies for semantic applications. TopBraid Composer offers extensive support for developing, managing of knowledge model engineering.

I benchmarked the proposed solution with those existing applications using five criteria: Semantic Knowledge Requirements, Ontology Discovery, Ontology Management, Data Presentation, UX, and Setup Effort. Semantic Knowledge Requirements are the general levels of understanding of ontology that users need to have in order to use the applications effectively. Ontology Discovery and Ontology Management consider the abilities of the tools to help users browsing existing ontology base and managing their ontology,

including edit and delete functionalities. Data Presentation evaluates how the tools present data models to users. UX is for the general usability and intuitiveness of the tools. Setup Efforts assesses the installation overhead needed to start using the tools.

Protégé is the most common open-source ontology editor on the market. It claims to be suitable for both beginners and expert users. However, I found it more on the expert side. The main view of Protégé does not display the relevant data for most of the significant use cases, but only shows the metrics of the ontology. The entities view of Protégé clearly shows the entities in a hierarchical structure format; however, the list is static, and there are no immediate ways to edit or add new classes or properties. By clicking to an entity on the list, users can open the individual entity screen, which shows all details of the selected entity. The information shown is useful; however there are many tabs and with no clear instructions. For users with less technical experience, this could be confusing. Editing information can be made inline, which is very straightforward. All inputs are equipped with auto-complete functionality. However, it requires a solid understanding of ontology engineering to create and edit data models since the tool does not provide any bits of help or instructions for starters. On the desktop version, there are several plugins providing ontology visualisation. However, they are only able to create a simple graph representation of data without any editing functionalities. Users can not see the details of each entity or make any changes on the graph. On the web version, ontology visualisation is not available.

TopBraid Composer is a refined, enterprise-class application, so it expects users to be at expert levels. The application does not provide any instructions on the UI. On the main view, TopBraid Composer displays useful information and an excessive list of buttons without any descriptions about their functionalities. This would be overwhelming for non-expert users. The list of entities view and single entity view are similar and comparable to Protégé. There are no significant differences; however TopBraid Composer offers code view, which shows the entity's serialisation in various formats, including RDF/XML, Turtle and JSON-LD, while Protégé lacks this feature. Data visualisation is only available for the paid version. Installation is required and a web version is not available.

NeOn-Toolkit comes in two versions: a basic configuration which provides basic OWL and F-logic ontology modelling functionalities, and an extended configuration which contains plugins, enabling more sophisticated features

such as external data integration, ontology mapping and reasoning. For both versions, installations are required, and a web deployment of the application is not available. NeOn-Toolkit offers similar but simpler functionalities to Protégé. It also has an ontology navigator, which shows a list of entities in a hierarchical structure, and an entity editor form for an individual entity. The features and user interfaces between two applications are comparable, and there are no significant differences. Users also are expected to have decent ontology engineering knowledge to use the tool. NeOn-Toolkit also provides simple data visualisation via plugins; however, data manipulation features on the graph are not available.

The assessment table is shown below.

	Protégé	**NeOn-Toolkit**	**TopBraid Composer**	**The proposed solution**
Semantic Knowledge Requirements	Basic knowledge of ontology engineering is required	Clean and fast for average users	For expert users with a solid understanding of ontology and triples	Both non-expert users and experts users can interact with the tool effortlessly.
Discovery	High discoverability. All inputs are auto-completed, and suggestions are displayed with details	High discoverability. All inputs are auto-completed, and suggestions are displayed with details	High discoverability. All inputs are auto-completed, and suggestions are displayed with details Provides code view, which shows data serialisation in RDF/XML, Turtle and JSON-LD	All inputs are auto-complete with quick actions. Users can choose to use, copy, or extend the concept based on their needs.
Ontology Management	Entities are displayed in a hierarchical structure. However, no quick ways to edit or add new classes or properties are supported.	Entities are displayed in a hierarchical structure. Data editing on the list is not supported.	Entities are displayed in a hierarchical structure.	Entities are displayed in a hierarchical structure. Users can make changes directly on the list view.
Data presentation	Simple data visualisation is available via plugins. Missing individual entity details and on-graph data editing features	Supports static data presentation. Direct data editing features are unavailable	Only available in the paid version	Ontologies are visualised on the graph with intuitive, drag and drop editing features.
Intuitiveness/ UX	Uses form-based structure, suitable for more experienced users	Uses form-based structure, suitable for more experienced users	Uses form-based structure, suitable for more experienced users	Provides a combination of self-explanatory forms and interactive graph, which allow users of any levels to create and edit ontology effortlessly
Setup Effort	There are two versions of Protégé. The desktop version is more powerful but requires more offline installation and more computing resources. The other one isWebProtege, which is a compact version but requires no installation.	Installation is required.	Installation is required.	No installation is required. The application works seamlessly in most major browser environments

Table 5.2: Comparison table between the proposed solution and existing tools

Overall, unlike many existing ontology editing tools, which are mainly for expert users who have semantic expertise to read and write triples, our proposed solution offers an intuitive way for users with fewer experiences with semantic engineering to create and manipulate data models effortlessly. We also provide a visualisation of ontology on an interactive graph, which is not standard on any other existing tools.

Chapter 6

Conclusions

With the dispersion of devices and sensors in the IoT industry, data interoperability has emerged as a burning question. The current IoT infrastructure and communication protocols make it is difficult for organisations to share and exchange IoT data, which is often available in vastly diversified formats. Besides, existing ontology engineering software only provide users with limited tools to specify the high-level description of data. Therefore IoT data usually contains inconsistencies, non-regulated naming conventions and terms. This work proposes to apply semantic interoperability into IoT landscapes. Semantic technologies allow data to be embedded with its meaning in a hierarchical structure. This helps promote high-quality, self-explanatory, reusable data models, which enables organisations to share and exchange IoT data to generate business value effortlessly. In this thesis, we have presented a web-based ontology editing application for IoT networks.

We first reviewed the concept and the applications of IoT, which cover a wide range of industries and everyday life activities. We discussed the current state of the art of data interoperability in IoT and how applying ontology can help cope with the current situation. While ontology can give meanings IoT data, the task of building, storing and sharing data models can be expensive and time-consuming. We then proposed a web-based graphical user interface for designing and sharing data models. The primary objective of the thesis work is to implement an ontology editor that allows any users, including both users with and without triple engineering knowledge, to create and edit data models effectively. There are two major requirements for the data designer:

- **Minimal setup effort**: the tool must require minimal-to-none software setup effort. We proposed a web-based solution that works in all major browsers without any setup overhead.
- **Ontology usability and interoperability**: the tool must promote

data model usability and sharing.

We surveyed several available open-source libraries to develop the data designer application. The front-end implementation of the application was built on React.js, which is a modern, component-centric JavaScript library for building user interfaces. The backend was a RESTful Python-Tornado implementation. The application is constructed as a single page web application that utilises AJAX and REST APIs technologies to provide users with engaging user experience. With a graph-based approach, the usability has been enhanced significantly with new features and more intuitive UI/UX. The performance of the application has also been improved noticeably. A new major feature that was introduced in this thesis work is the support for building multi-dimensional data models. Moreover, the smart input system recommends users to reuse existing ontologies that may match their need when they are creating new data models. This helps to encourage the reusability and sharing of data models in the system.

Overall, we fulfilled all requirements and specifications, and the application was working properly on different browser environments. We evaluated the application on various factors using expert opinions and user group survey. In general, both users with ontology engineering knowledge and non-technical users could perform different tasks timely and effortlessly. We collected the feedback and iterated the work accordingly. Although there were still rooms for improvement in the application, such as a new view for uploading ontology or a better organisation of class hierarchical tree, the application was robust and working as designed.

We also benchmarked our application with the existing tools on the market. The current tools are apparently for ontology engineers who are already familiar with creating and manipulating triples. They are usually very confusing and hard to use for non-expert users. Our application would fit into that gap. By providing an intuitive graphical user interface with a graph that visualises data models and their relations, Data Designer can help ease the process of creating and managing ontologies.

Furthermore, the solution showed promising potential for further development and expansion. Our ideas for future works are presented below:

- Support Turtle/XML as input for data models: for more technical users, creating data models directly using Turle or other RDF serialisation syntaxes would be more productive.

- Code Generator add-on/extension for major IDEs: we have thoughts of building extensions that generate code stubs for major code editing

tools, such as Vim, Atom, VsCode. When users are building triples in those IDEs, the extension can read the ontology and generate matching code stubs, similar to the code generator in the Data Designer.

- Better Discovery Service: in the current version, the smart input provides users with suggestions upon their input keywords. However, we would want to build a map of existing ontologies so that users can discover and fetch what they are looking for more easily.
- Enterprise Integration: one major improvement to the application is the ability to integrate with existing IoT infrastructures.

Bibliography

[1] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, October 2009.

[2] Simona Enescu. The concept of cybersecurity culture. *The Fourth Annual Conference of the National Defence College Romania in the New International Security Dynamics*, pages 176–191, 2019.

[3] Yuan Li, Mingjun Hou, Heng Liu, and Yi Liu. Towards a theoretical framework of strategic decision, supporting capability and information sharing under the context of internet of things. *Information Technology and Management*, 13(4):205–216, 2012.

[4] B. Ahlgren, M. Hidell, and E. C. . Ngai. Internet of things for smart cities: Interoperability and open data. *IEEE Internet Computing*, 20(6):52–56, Nov 2016.

[5] Payam Barnaghi, Wei Wang, Cory Henson, and Kerry Taylor. Semantics for the internet of things: Early progress and back to the future. *Int. J. Semant. Web Inf. Syst.*, 8(1):1–21, January 2012.

[6] Joshua Cooper and Anne James. Challenges for database management in the internet of things. *IETE Technical Review*, 26(5):320–329, 2009.

[7] J. Zheng, D. Simplot-Ryl, C. Bisdikian, and H. T. Mouftah. The internet of things [guest editorial]. *IEEE Communications Magazine*, 49(11):30–31, November 2011.

[8] L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014.

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

[10] Chunling Sun. Application of rfid technology for logistics on internet of things. *AASRI Procedia*, 1:106 – 111, 2012. AASRI Conference on Computational Intelligence and Bioinformatics.

[11] Dhruv Grewal, Anne L. Roggeveen, and Jens Nordfält. The future of retailing. *Journal of Retailing*, 93(1):1 – 6, 2017. The Future of Retailing.

[12] Yuehong YIN, Yan Zeng, Xing Chen, and Yuanjie Fan. The internet of things in healthcare: An overview. *Journal of Industrial Information Integration*, 1:3 – 13, 2016.

[13] Xiang Su, Jukka Riekki, Jukka K. Nurminen, Johanna Nieminen, and Markus Koskimies. Adding semantics to internet of things. *Concurrency and Computation: Practice and Experience*, 27(8):1844–1860, 2015.

[14] AV Dastjerdi R Buyya. *Internet of Things: Principles and Paradigms*. Morgan Kaufmann, 2016.

[15] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684 – 700, 2016.

[16] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431 – 440, 2015.

[17] X. Jia, Q. Feng, T. Fan, and Q. Lei. Rfid technology and its applications in internet of things (iot). In *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1282–1285, April 2012.

[18] L. Mainetti, L. Patrono, and A. Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pages 1–6, Sep. 2011.

[19] Kiev Gama, Lionel Touseau, and Didier Donsez. Combining heterogeneous service technologies for building an internet of things middleware. *Computer Communications*, 35(4):405 – 417, 2012.

[20] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[21] Pankesh Patel, Animesh Pathak, Thiago Teixeira, and Valérie Issarny. Towards application development for the internet of things. In *Proceedings of the 8th Middleware Doctoral Symposium*, MDS '11, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[22] Sarfraz Alam, Mohammad M. R. Chowdhury, and Josef Noll. Interoperability of security-enabled internet of things. *Wireless Personal Communications*, 61(3):567–586, Dec 2011.

[23] B. Di Martino, A. Esposito, S. A. Maisto, and S. Nacchia. A semantic iot framework to support restful devices' api interoperability. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, pages 78–83, May 2017.

[24] S. Zawoad and R. Hasan. Faiot: Towards building a forensics aware eco system for the internet of things. In *2015 IEEE International Conference on Services Computing*, pages 279–284, June 2015.

[25] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson. M2m: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43, April 2011.

[26] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. Om2m: Extensible etsi-compliant m2m service platform with self-configuration capability. *Procedia Computer Science*, 32:1079 – 1086, 2014. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

[27] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the internet of things. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sep. 2015.

[28] Linux Foundation AllSeen Alliance. Alljoy wiki, 2016.

[29] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, and Katarzyna Wasielewska. Semantic interoperability in the internet of

things: An overview from the inter-iot perspective. *Journal of Network and Computer Applications*, 81:111 – 124, 2017.

[30] F. van Harmelen R. Hoekstra G. Antoniou, P. Groth. *A Semantic Web Primer. 3rd Edition.* MIT Press, 2012.

[31] Tim Berners-Lee. Linked data, 2006.

[32] W3C. 5 star linked data, 2013.

[33] Li Ding, Tim Finin, Anupam Joshi, Yun Peng, Rong Pan, and Pavan Reddivari. Search on the semantic web. *Computer*, 38:62 – 69, 11 2005.

[34] C. Bizer T. Heath. *Linked Data: Evolving the Web into a Global Data Space.* Morgan Claypool, 2011.

[35] W3C. An introduction to multilingual web addresses, 2008.

[36] Asema Oy. Smart api core ontology model, 2018.

[37] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and S Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24:45–77, 01 2007.

[38] Jabu Mtsweni, Elmarie Biermann, and Laurette Pretorius. isemserv: A model-driven approach for developing semantic web services. *South African Computer Journal*, 52, 06 2014.

[39] Asema Oy. Asema iot central architecture and operation, 2018.

[40] Standford University. Protégé.

[41] Neon-Toolkit. Neon toolkit wiki, 2014.

[42] W3C. Topbraid wiki, 2011.