

Mobile Assets in Semantic Digital Twins

Oscar Lund Ramstad



Thesis submitted for the degree of
Master in Informatics: Programming and System
Architecture
30 credits

Institute for Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2023

Mobile Assets in Semantic Digital Twins

Oscar Lund Ramstad

© 2023 Oscar Lund Ramstad

Mobile Assets in Semantic Digital Twins

<http://www.duo.uio.no/>

Printed: Reprosentralen, University of Oslo

Abstract

This thesis proposes a solution to the challenge of handling mobile assets in a semantic digital twin (DT) based on a dynamic asset model in which static data (existing building infrastructure) is separated from dynamic data (physical location of mobile assets). The DT is an up-to-date digital representation of its physical twin (PT), and they communicate both ways. The DT easily integrates with semantic technologies and is enriched by them. A cross-platform app is created for smartphone users to send their status, whilst protecting the privacy of data subjects as much as possible. If a person is in danger, he is warned in the app from an informed decision sent by the DT. The conclusion of being inside a critical area is drawn from sufficient data. This was evaluated to be the case for 234 smartphones. We provide informative messages in a SPARQL endpoint, enabled by the SMOL REPL. The proposal is extensible and inexpensive to maintain by manual operations. Formalized knowledge of the domain is ensured when automatically reading from a knowledge graph, even if the set of heterogeneous smartphones changes.

Acknowledgements

I would like to extend my greatest gratitude to those that helped me with this thesis. Thanks to Einar Broch Johnsen, Rudolf Schlatte, and Eduard Kamburjan for formidable guidance in writing this thesis. Einar, thanks for believing in me from the start. I have learned a lot from the discussions on separating the static analysis from the dynamic snapshots. Rudi, I have thoroughly enjoyed our many discussions on relevant subjects to this thesis, especially about the building domain and privacy. Eduard, thanks for giving me great feedback on my writing and everything SMOL related.

Thanks to my family, and especially my parents, for being there for me no matter what. I could never have done this without any of you.

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Problem Statement	2
1.4	Thesis Scope and Outline	3
2	Background	5
2.1	Digital Twin	5
2.1.1	Historic Background	5
2.1.2	Definition	5
2.2	Semantic Web	6
2.2.1	Technologies	6
2.3	Asset Model	8
2.4	SMOL	10
2.4.1	SMOL Interpreter	10
2.5	Reasoning	10
2.5.1	HermiT	11
2.6	App Development	12
2.7	Databases	13
3	Problem analysis and design	15
3.1	Research Questions (RQ1, RQ2, RQ3)	15
3.1.1	RQ1	15
3.1.2	RQ2	15
3.1.3	RQ3	16
3.2	Building Domain	16
3.3	Dynamic Asset Model	17
3.4	Data Separation	18
3.4.1	Static and Dynamic Data	18
3.4.2	Further Data Separation	19
3.5	Analysis of Technologies	19
3.5.1	App Development	19
3.5.2	Databases	20
3.5.3	Semantic Digital Twin	20
3.6	Structural Requirements	21
3.7	Design	22
4	Implementation	23
4.1	Prototypical Implementation	23
4.2	Implemented Solution	25
4.2.1	Client	25
4.2.2	Server	27
4.2.3	Databases	30
4.2.4	Twin	30

4.3	Implemented Requirements	33
5	Evaluation	35
5.1	Setup of SMOL	35
5.2	Guided Test Runs	35
5.2.1	Automatically Reloading Asset Model	35
5.2.2	Manually Adding Smartphones	39
5.3	Manually Warning	40
5.3.1	Goal	40
5.3.2	Experimental Setup	40
5.3.3	Discussion of Results	40
5.4	Many smartphones	41
5.4.1	Goal	41
5.4.2	Experimental Setup	41
5.4.3	Discussion of Results	41
6	Discussion	44
6.1	Dynamic Data	44
6.2	Data in the Twin	44
6.3	Client-server Communication	45
6.4	Dimensions and Areas	45
6.5	Privacy	46
6.6	Discussion of Implementation	46
6.7	Discussion of Contributions	47
7	Conclusion and Further Work	48
7.1	Related Work	48
7.2	Contributions	49
7.3	Future Work	50

List of Figures

1	Overview of main architectural components	3
2	A simple building with the structural components: two rooms (R1 and R2) connected by an inner wall (W) with a door (D) in it	4
3	RDF triple consisting of a subject, a predicate, and an object	7
4	RDF graph consisting of two triples in its set	7
5	Ontology of a simple building with the object properties (hasDoor, hasWallLeft, hasWallRight), the classes (Room, Wall, Door), and the individuals (D, R1, R2, W)	9
6	A simple SMOL program (example.smol)	10
7	Reasoning over an ontology in Protégé with HermiT	11
8	Accessing time-series data from InfluxDB in a function written in SMOL	14
9	A UML class diagram that illustrates the separation of static and dynamic classes	17
10	The separation of static and dynamic data illustrated in blue and red, respectively	18
11	Wireframe of the UI in the app	22
12	Overview of architectural components in the final implementation	23
13	Physical location recording is <i>off</i> (green button)	24
14	Physical location recording is <i>on</i> (red button)	24
15	The color theme in the app	26
16	Rendering of the location recording button in the app	26
17	The mobile asset (smartphone) in the PT is <i>safe</i> , as concluded by the DT	27
18	<i>Endangered</i> smartphone from the informed decision to be inside a critical area	27
19	Sending its status to Firebase	27
20	Get updated sensor data from Firebase	28
21	Forward informed decision to PT through Firebase	29
22	Inserting a measurement point (timestamped status message) into the Data-bucket in InfluxDB	29
23	Accessing the updated sensor data from InfluxDB	31
24	Creating a smartphone object from the smartphone ID in the asset model, in combination with the corresponding updated coordinates from InfluxDB	31
25	Checking if a smartphone is inside an area programmatically	32
26	Checking if a smartphone is inside an area that is set to critical. If the area contains its point (x, y), the smartphone is set to endangered, and is added to the list for the specific area	32
27	Querying with SPARQL	33
28	Server detects a smartphone	36
29	Time-series data in InfluxDB	36
30	Time-series data represented in a map in InfluxDB from using a Flux query	37

31	Smartphone individual with ID 1 is the only change in the file . .	38
32	Informative messages in the REPL from running the SMOL program in the DT	38
33	Printing any smartphones without a physical location	39
34	Map showing 234 smartphones	42
35	All smartphones are endangered	42
36	None are safe, which means all are endangered	43
37	Shows to what extent data can be considered dynamic	44

1 Introduction

1.1 Context

A digital twin (DT) is a digital representation of some physical system, referred to as a physical twin (PT), that is twinned in near real-time by the DT in order to understand or control the PT [17, 31]. The twinning aims to be bidirectional meaning that the DT both observes the PT and communicates informed decisions back to it [31, 12]. The DT must continually adapt to correctly reflect its physical twin [33]. This may pose a challenge if the PT contains mobile assets, such as smartphones. The number of smartphone users worldwide is at an all-time high [44]. They are important resources for a company and can change at any time [41].

It is stated by Godager, Onstein, and Huang that although Building Information Modeling (BIM) could serve as a basis for DTs by providing lots of relevant information, DTs should provide more context about the building, especially by monitoring assets from near real-time data. A lot of work goes into modeling physical assets in general [62], and using BIM is also very costly due to frequent updates by maintainers [20].

Knowledge graphs (i.e. using graphs to represent data from diverse sources [22, 49]) are used in the interconnection of various elements in DTs. Furthermore, instead of storing all related data in one place, Waszak et al. propose storing static and dynamic data sources in different places and linking them.

There are challenges in handling mobile assets. Some systems mainly focus on tracking mobile assets and the accuracy of their physical location [41, 4]. If we keep the company’s assets in mind, such a system could be improved upon by better understanding and controlling asset behavior.

Furthermore, there is existing research about using Building Information Modeling (BIM) with standard data formats, such as JSON or Resource Description Framework (RDF), to better define static building infrastructure [43]. Pauwels et al. proposes how, and to what extent, such data can be used in robot navigation. Although BIM can be used, by default it has some challenges regarding interoperability due to heterogeneous data [10, 16].

Handling mobile assets in a DT based on an asset model (i.e. a file that contains an organized description of an asset’s composition and properties [33]) that is *dynamic* is unexplored territory.

1.2 Motivation

Given the gap that exists in connecting tracking systems, asset models, and digital twins, there is still room for improvement. The problem is that there exist no systems that have a digital twin (DT) based on a dynamic asset model which can track an unlimited number of heterogeneous smartphones. In an attempt to formalize heterogeneous data and make it understood by computers, semantic web technologies should be used. More specifically, we aim to semantically enrich a model of a building to increase interoperability, as well as keep it

inexpensive and easy to maintain.

We envision an extension of a semantic digital twin (i.e. a digital twin that processes heterogeneous data from different sources in the domain into generally understandable information [8]) that handles mobile assets, and is based on a dynamic asset model in which static data (existing building infrastructure) and dynamic data (smartphone's physical location) is separated. The tracking system will track an unlimited number of mobile assets simultaneously. The DT will be formalized using semantic web technologies. In the knowledge graph, static data will be noted to be different than dynamic data as it is generated from a combination of the program state and the asset model.

The asset model will be automatically reloaded by the system itself, or manually updated by an operator either offline (online version updated from loading local changes), or directly online in an ontology editor. In this way the mobile assets are not only tracked, but also twinned in near-real time by the DT. It gets sensor data from the PT, and sends informed decisions back to it.

1.3 Problem Statement

In the context of digital twins and an increasing number of smartphone users, as well as the motivation for improvement described in Section 1.2, we present the following hypothesis (H):

H: We can handle mobile assets in a semantic digital twin with the use of a dynamic asset model of a simple building, in which dynamic data (smartphone's physical location) is automatically updated by a server and separated from static data (existing building infrastructure), and informed decisions are sent back to the physical twin.

In addition to this, we present the following three research questions which will guide this thesis:

- RQ1:** How to create a dynamic asset model of a simple building that is also extensible?
- RQ2:** How to enable bidirectional data flow between the PT and DT, so that the digital twin gets updated sensor data from mobile devices and sends informed decisions back?
- RQ3:** How to separate dynamic data (smartphone's physical location) from static data (building infrastructure) in the asset model?

Having the above points in mind, an overview of the main architectural components in this thesis is shown below in Figure 1.

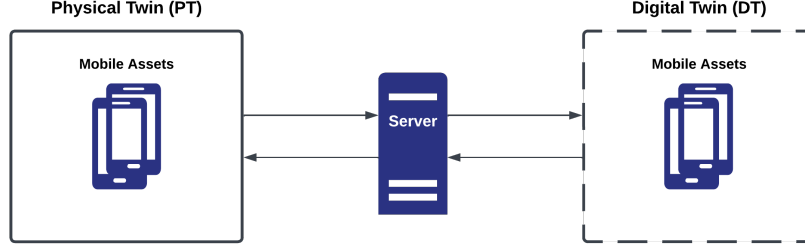


Figure 1: Overview of main architectural components

1.4 Thesis Scope and Outline

The scope of this thesis is to handle many mobile assets in a semantic digital twin based on a dynamic asset model. Figure 1 shows the overview of architectural components and the bidirectional data flow between the PT and DT. The digital twin should be developed using the Semantic Micro Object Language (SMOL) described in Section 2.4 and its publicly available interpreter¹.

As noted by Pauwels et al. many physical objects, such as furniture and doors, can be movable entities in a building apart from the robots [43]. Doors will have one state (open). Laptops can also be considered mobile assets, but due to the context of a crisis situation, we are not able to justify adding support for it. We limit ourselves to smartphones and tablets in 2D space.

The areas are predefined squares with latitude and longitude ranges, where we assume the coordinates align. Square areas are easier to compute and scale into larger areas, and we are not concerned with other shapes such as polygons. The analysis of what certain areas are (e.g. what does critical areas mean) is also outside the scope of this thesis.

We also assume the PT contains a building in which there can be smartphones since we create a digital representation of a building in the DT.

Figure 2 shows a simple building with two rooms connected by an inner wall with a door (open) in it.

¹<https://github.com/smolang/SemanticObjects>

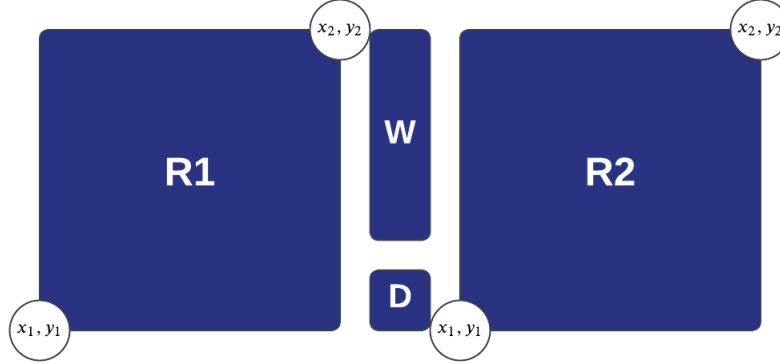


Figure 2: A simple building with the structural components: two rooms (R1 and R2) connected by an inner wall (W) with a door (D) in it

The remainder of this thesis is organized as follows:

- Chapter 2 describes the theoretical background of this thesis.
- Chapter 3 investigates the problem statement and the research questions. We also look at what a dynamic asset model means, as well as the separation of static and dynamic data.
- Chapter 4 dives into the implementation of the proposed solution and how the different architectural components shown in Figure 12 interact. More specifically, how do we "close the loop" by sending informed decisions back to the PT.
- Chapter 5 evaluates the proposed solution according to the structural requirements that were set, and then according to various test runs.
- Chapter 6 discusses the limitations of the discoveries in this thesis, the alternatives, and compares them to other existing work.
- Chapter 7 concludes this thesis and presents the contributions. We look at improvements that can be made to the proposed solution, and further work that can be done using the thesis as a basis.

2 Background

This chapter describes the background information of this thesis. Starting off we give a description of digital twins (DTs). Then we give a description of the Semantic Web, as well as the technologies that help us build it. Later on we describe semantic reasoning and tools for it. Then we look at frameworks for building cross-platform applications from a single codebase. Near the end of the section, we describe real-time and time-series databases.

2.1 Digital Twin

2.1.1 Historic Background

The *idea* of a digital twin can be traced back to the 1960s in NASA’s space exploration days, where multiple simulations were employed to evaluate the problems with the space mission Apollo 13 [9, 12]. The *term* digital twin was first introduced to the public by Michael Grieves in a presentation given in 2002, and in 2014 conceptualized as a virtual representation of a factory replication [18]. This set the foundation for developing digital twins [18, 12]. Since then, DTs have had many applications in various disciplines, and in recent years especially in smart cities, healthcare, manufacturing, and asset management [12, 62, 38]

2.1.2 Definition

As of today, a DT is referred to as a digital representation of some physical system [17], that has no restriction on the size or scale of it, which means the entity could be a single asset or a system of systems for that matter [37, 62]. The physical system exists in real life and is commonly referred to as the physical twin (PT), and is observed by the DT in near real-time. There should be a bidirectional flow of data between the DT and the PT throughout the life cycle of the PT. In other words, the DT gets updated sensor data from the PT, and sends informed decisions back to it [39, 62, 31]. From this, and *for the scope of this thesis*, a DT is a digital representation of a system that tracks many mobile assets. The DT sends informed decisions to the physical counterpart in which the appropriate mobile assets perform actions.

There are multiple misconceptions about DTs. One such misconception is that a digital shadow (DS) is a DT. This misconception often leads people to build shadows instead of DTs. [12, 37]. Although very similar, a digital shadow (DS) does not have the bidirectional data flow that a DT does. Instead, there is a one-way flow from the PT to the digital counterpart, which often leads to a change in the DS, but not vice versa [35, 37]. Secondly, as told by Fuller et al., a common misconception is that the DT must be an exact 3D copy of a physical entity, such as a building.

In order to capture diverse data and formalize knowledge represented in DTs, knowledge graphs should be used [32, 62]. Waszak et al. created a DT architecture in which static and dynamic data sources were not explicitly stored

in a common knowledge graph but instead linked to one another for better scaling and heterogeneity.

To sum up, it is important to understand what a DT is and what it is not. We should try to "close the loop" by sending informed decisions back to the physical counterpart so that it reflects the newest state of the DT. A DS could be used alongside the DT with the only goal of being a timestamped snapshot of some specific physical asset [7]. When implementing one should also think about the challenges of digital twins. We should not get too caught up with modeling an exact 3D replica of an asset. On the other hand, we should use existing semantic technologies, such as knowledge graphs, to formally represent knowledge from a variety of data sources.

2.2 Semantic Web

Semantic Web was envisioned as an extension of the World Wide Web (WWW) in which data was linked. Linked data is about making links so people and machines can explore the web of data, and according to Tim Berners-Lee, should follow the four rules [59] below:

1. Uniform Resource Identifiers (URIs) should be used as names for things.
2. HTTP URIs should be used so that people can look up the names.
3. Useful information should be provided when people look up URIs, by using RDF or SPARQL.
4. Links to other URIs should also be provided so that people can discover more things.

Later, these four rules evolved into five rules for linked *open* data (i.e. linked data that can be freely used and distributed) [1].

2.2.1 Technologies

There are foundational semantic technologies and standards that facilitate data exchange and data integration on the web. The idea is to give meaning (semantics) to web resources in a format that can be processed by computers. Computers should also be able to access more of the information that previously required human attention and time [21].

In order to create an ideal future Web of linked data where computers can access more information based on what meaning this content has to humans, World Wide Web Consortium (W3C) has set forth some Semantic Web technologies [52], which are briefly described below:

- **RDF** is a data model for describing (meta)data and its interchange on the web. Its linking structure forms a directed, labeled graph (RDF graph) [50, 54]. An RDF graph is a set of RDF triples. Triples consist of a subject, a predicate, and an object, as shown in Figure 3. They can be

thought of as facts. By using them, structured metadata can be exposed and shared between different applications [48]. Triples are typically stored in triple stores (databases purposely built for storing triples) and can be retrieved with semantic queries [61].

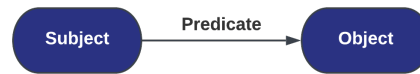


Figure 3: RDF triple consisting of a subject, a predicate, and an object

Figure 4 shows an RDF graph with two triples in its set. Both triple's subject is the URL of a web page. One triple also has the predicate **dc:title** and the object **Building**, whilst the other one has the predicate **dc:description** and the object **An ontology of a simple building**. As we can see, each of the RDF terms (subject-predicate-object) can be e.g. URLs or string literals.

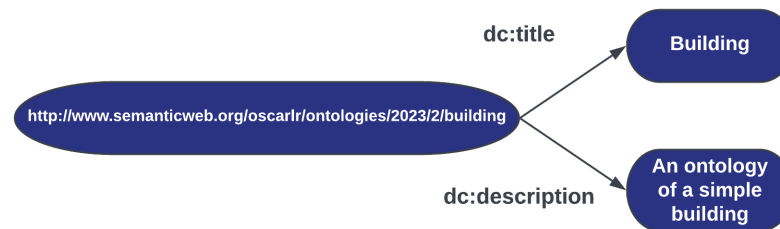


Figure 4: RDF graph consisting of two triples in its set

- **SPARQL** is a query language for RDF and is defined by W3C. It expresses queries for varying sources of data, and the results from querying are sets of results or RDF graphs [54].

Below is an example of querying the title (**Building**) in the RDF graph shown in Figure 4.

Data:

```

<http://www.semanticweb.org/oscarlr/ontologies/2023/2/building>
<dc:title>
"Building"
  
```

Query:

```

SELECT ?title
WHERE
{
  <http://www.semanticweb.org/oscarlr/ontologies/2023/2/building>
  <dc:title>
  ?title .
}

```

Result:

title
"Building"

Table 1: Query result

- **Web Ontology Language (OWL)** is a language for expressing rich knowledge of concepts (phrases in natural language) in an application domain [55], as well as relations between entities. More specifically, ontologies describe domains in terms of classes, properties, and individuals [6].
- **Shapes Constraint Language (SHACL)** is a language for validating an RDF graph against a set of conditions as shapes expressed in another RDF graph. It is said that "data graphs" can be validated against "shapes graphs" [53].
- **JSON-LD** JSON-LD is a JSON-based format for encoding and serializing linked data. Although not a well-established standard, it is still defined by W3C. More specifically, it enables JSON objects to contain semantic links [3].

Commonly, these technologies can be used to create semantic [digital twins](#).

2.3 Asset Model

Although BIM (Building Information Modeling) can be used with RDF and JSON to create static building infrastructure as mentioned in Section 1, some meaning is lost in translation due to a lack of semantic enrichment. BIM can be used in DTs for more context about the built environment which includes asset behavior [16]. However, a digital twin should aim to have interoperability between heterogeneous data sources.

OWL documents can easily be created (from exporting), loaded into, and maintained in ontology editors, such as Protégé². Such a tool can be used to give semantics to e.g. a model of a building, as well as spending less time on creating and maintaining it compared to BIM.

²<https://protege.stanford.edu>

Figure 5 shows an example of an ontology of the simple building shown in Figure 2 in Turtle Syntax (i.e. data format for RDF data model [56]). This ontology uses semantic technologies outlined in Section 2.2.1, such as RDF, which again is used by OWL to formalise knowledge in this specific building domain. For a more compact view of the ontology, URLs are removed and the coordinates of the room border corners are purposely left out, and some white space and comments are also removed. We also do not take directions into account meaning the room R1 will always be left of R2, and R2 always to the right of R1. Note that this is just an example of an ontology and that a real building domain would be many times more complex.

According to W3C’s documentation, RDF graphs (such as the one shown in Figure 5) are static snapshots of information, but by giving appropriate vocabulary collections of Internationalized Resource Identifiers (IRIs) [47], observations about entities or groups of entities through time can be captured.

Lastly, this data in the ontology can also be reasoned over with technologies outlined in Section 2.5, which contains an example of reasoning over an inconsistent ontology in Protégé (desktop version), as well as a description on how to make the ontology consistent.

```
ex:hasDoor rdf:type owl:ObjectProperty .
ex:hasWallLeft rdf:type owl:ObjectProperty .
ex:hasWallRight rdf:type owl:ObjectProperty .

ex:Door rdf:type owl:Class .
ex:Room rdf:type owl:Class .
ex:Wall rdf:type owl:Class .

ex:D rdf:type owl:NamedIndividual ,
      :Door .

ex:R1 rdf:type owl:NamedIndividual ,
      :Room ;
      :hasWallRight :W .

ex:R2 rdf:type owl:NamedIndividual ,
      :Room ;
      :hasWallLeft :W .

ex:W rdf:type owl:NamedIndividual ,
      :Wall ;
      :hasDoor :D .
```

Figure 5: Ontology of a simple building with the object properties (hasDoor, hasWallLeft, hasWallRight), the classes (Room, Wall, Door), and the individuals (D, R1, R2, W)

2.4 SMOL

SMOL is an imperative, object-oriented research language [58]. According to the introduction of the SMOL language, it integrates semantic technologies, and can be used as a framework for creating DTs. For these DTs, the knowledge graphs can be used to capture asset models [29]. The source code of SMOL is publicly available³. What follows is a simple SMOL program that creates a smartphone object and prints **Careful!** if a criterion is met:

```
class Smartphone(Int id, String name, Boolean isBlackBerry) end
main
  Boolean isInsideRaspberryField = True;
  Smartphone blackberry = new Smartphone(1, "Evolve", True);

  if (blackberry.isBlackBerry & isInsideRaspberryField) then
    print("Careful!");
  end
end
```

Figure 6: A simple SMOL program (example.smol)

2.4.1 SMOL Interpreter

The interpreter reads and executes the SMOL program in a given file. The example program (in the file **example.smol**) shown in Figure 6 was executed in the terminal. Note that the second and fourth line starts with **M0>**. This simply means commands are executed inside the REPL.

```
java -jar smol.jar
M0> read example.smol
Careful!
M0>
```

2.5 Reasoning

OWL was previously mentioned in Section 2.2.1 as a language for expressing rich knowledge of concepts and relations between entities. We should keep in mind, however, that this formalized knowledge should be reasoned over and put in context to draw conclusions based on criteria. There are many technologies that let us do this.

³<https://github.com/smolang/SemanticObjects>

2.5.1 HermiT

HermiT is a reasoner for OWL 2 [15], which is an upgraded version of OWL. Glimm et al. describes that the reasoner support both object and data property classification, as well as SPARQL query answering [15]. HermiT is built into the ontology editor Protégé and can be used to reason over an ontology directly in the editor. Other plugins for reasoning can be added to the ontology editor, but it is convenient that HermiT comes pre-installed with the desktop version.

Figure 7 shows an ontology that is *inconsistent* (i.e. an ontology that cannot have any models and entails everything [24, 26]). The information window **Help for inconsistent ontologies** tells us that the OWL reasoner (HermiT) will no longer be able to provide any useful information about the ontology. The ontology is inconsistent because the object properties **hasWallLeft** and **hasWallRight** are disjoint (having no elements in common), and the individual **R1** has both object properties. This makes no sense because the room can't have the wall on its left side and at the same time have it on its right side. This could be fixed by simply removing the object property **hasWallLeft** from the individual according to the context. The effect of an inconsistent ontology is that no meaningful conclusions can be drawn from it [24].

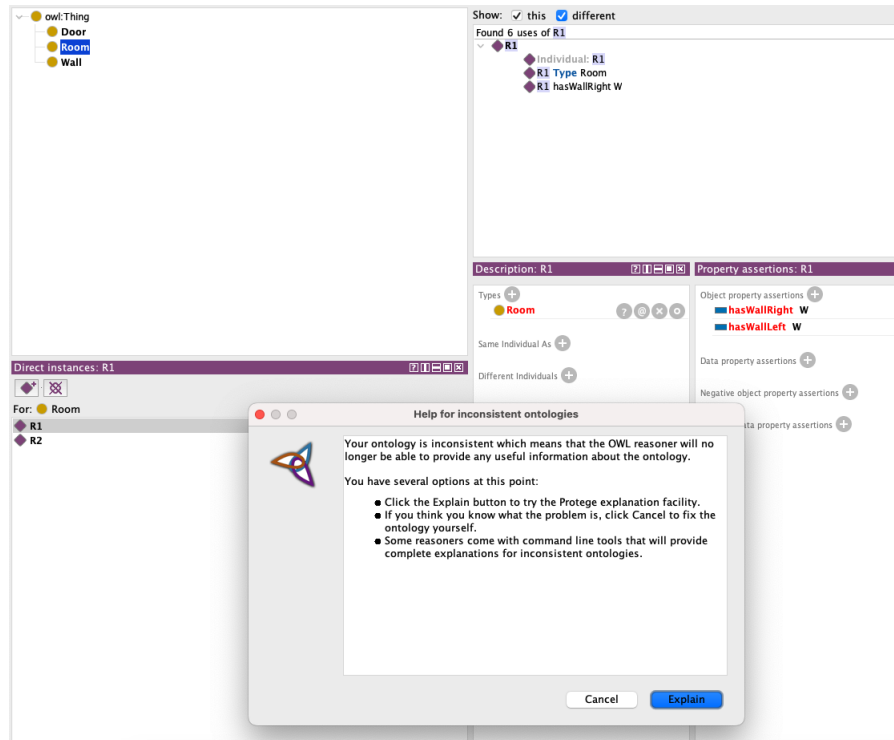


Figure 7: Reasoning over an ontology in Protégé with HermiT

There also exist other technologies that provide reasoner tools. Some notable mentions include, but are not limited to:

- **Apache Jena** is an open source framework that has the Inference API which include reasoner tools that can be used to reason over data, as well as checking content of triple stores [5]. The source code is publicly available⁴.
- **OWL API** is a Java framework that lets us create, serialize or manipulate ontologies written in OWL [57], and its source code is openly accessible⁵. Its recent development is more focused towards OWL 2 however.
- **SMOL** provide reasoning tools through its interpreter, which implements semantic lifting (generating a knowledge graph from the current state of a SMOL program) [51].

The knowledge graph is generated by using the `dump` command in the SMOL Read-Eval-Print Loop (REPL) (i.e. user inputs in a terminal are read and results are returned to the user). The knowledge graph is generated as an output file with the filename extension `.ttl`. Reasoning is done automatically by the interpreter whenever `access` or `member` are executed. Apache Jena reasoner is used for `access`, whilst `member` triggers reasoning over an OWL concept with HermiT. Validating with SHAQL does however not require reasoning.

2.6 App Development

We will describe the topic of app development by looking at concerns when developing in industry. Smartphones and tablets have different operating systems for mobile, in which the two most notable are iOS (iPhone and iPad) and Android (smartphones and tablets with different brands, such as Samsung, Google, and OnePlus). Smartphones are heterogeneous by nature. They not only have different operating systems but also have various device sensors, depending on the brand and price of the device.

One should develop for multiple platforms from a single codebase, provided that development time is crucial and that the app should have cross-platform (e.g. both iOS and Android) support. However, this should be planned early on in the development, preferably before anyone starts writing code. Some examples of frameworks that support cross-platform app development include:

- Flutter
- Ionic
- React Native

⁴<https://github.com/apache/jena>

⁵<https://github.com/owlcs/owlapi>

Devices are often equipped with a GPS (Global Positioning System), which gets signals from satellites to determine the GPS-based physical location of the device. Google provides a service, namely Google Location Accuracy, which collects additional location data from multiple sources, such as nearby Wi-Fi and cellular networks, to provide a more accurate physical location of a device [25]. The Google Maps SDK is a set of tools for integrating maps into applications, such as for Android and iOS. According to Google, some features of the Software Development Kit (SDK) are different map displays and map gesture responses [40].

Continuous tracking of physical devices raises privacy concerns. General Data Protection Regulation (GDPR)⁶ works to ensure the privacy and security of personal data. As an example, no personal data should be collected without the permission of the *data subject* (i.e. user), as stated in GDPR Article 13 and 14. For what purpose the data is collected should also be explicitly stated [2].

2.7 Databases

Databases are collections of information that exist for a long time, often over many years [13]. They can be divided into relational and non-relational (NoSQL) databases [42]. Relational databases have existed for many decades and store data in tables (which consist of columns and rows), whilst NoSQL databases store data differently. Examples of relational databases are MySQL and PostgreSQL, whereas examples of NoSQL databases are:

- **MongoDB** is a well-known NoSQL database for storing data in JSON documents and its source code is publicly available⁷.
- **Firebase Realtime Database** is a cloud-hosted NoSQL database that stores data as JSON objects [11]. Furthermore, data is synchronized in real-time to cross-platform clients.
- **InfluxDB** is a NoSQL database for storing time series data (i.e. "sequence of data points indexed in time order" [63]) [28], and its source code is also openly accessible⁸. Flux queries can be used to query time series data in the database.

According to Garcia-Molina et al., databases are managed by a Database Management System (DBMS), a system that should:

- Let users create new databases in which the logical structure of data (schema) is specified by the user using a data-definition language.
- Let users query and modify data with a query language.

⁶<https://gdpr-info.eu/>

⁷<https://github.com/mongodb/mongo>

⁸<https://github.com/influxdata/influxdb>

Non-relational variants have become increasingly popular in recent years due to how they solve scalability concerns with relational databases. NoSQL databases are of different types, such as document databases and graph databases. As described by Mohamed, Altrafi, and Ismail, traditional databases were not created with horizontal scaling (sharding) in mind [42]. More specifically, it does not do well with new machines being added to the pool of resources. Instead, it relies heavily on vertical scaling, which is adding more computing power (CPU, RAM) to an individual resource (machine) in the pool. On the other hand, NoSQL databases are optimized for scaling horizontally [42, 34].

We should consider that although databases can have many different uses, some are better for specific purposes than others. Time series databases can be integrated into digital twins (DTs) for getting updated sensor data from the physical twin (PTs). Section 2.4 describes that SMOL can be used as a framework to create DTs. Additionally, in SMOL we can query time series data from InfluxDB by using the `access` statement [60]. An example program of querying latitude and longitude coordinates from the start of the time range, and then returning the list of coordinates follows:

```
List<Double> getCoordinates()
    List<Double> coordinates = null;
    coordinates = access(
        "from(bucket: \"Data\")
        |> range(start: 0)
        |> filter(fn: (r) => r[\"_measurement\"] == \"data\")
        |> filter(fn: (r) => r[\"_field\"] == \"latitude\" or
        ↪ r[\"_field\"] == \"longitude\")
        |> aggregateWindow(every: 5m, fn: mean, createEmpty:
        ↪ false)
        |> yield(name: \"mean\")",
        INFLUXDB("influx.yml")
    );
    return coordinates;
end
```

Figure 8: Accessing time-series data from InfluxDB in a function written in SMOL

Firebase Realtime Database has multi-platform support for clients, such as support for Android and iOS devices [11].

3 Problem analysis and design

Given the gap that was identified in Section 1.2, the hypothesis (H) and its three guiding research questions (RQ1, RQ2, RQ3) described in Section 1.3 will in this chapter be examined based on the theoretical background information and general code examples in Section 2. We create a building domain based on the analysis of the research questions. Then we examine what a dynamic asset model entails, and how static data can be separated from dynamic data in it. Different technologies, such as databases, frameworks for creating applications (apps), and semantic digital twins are considered. Lastly, we present the resulting structural requirements as the basis for the implementation.

3.1 Research Questions (RQ1, RQ2, RQ3)

3.1.1 RQ1

RQ1 is concerned with creating a dynamic asset model of a simple building. From the descriptions of asset models in Section 2.3, an ontology of a simple building was created, as shown in Figure 5. This asset model is a simplified version of an exported OWL document (file). It can be created in an ontology editor, such as Protégé. Although other ontology editors can be used, we should use Protégé due to its user-friendliness and the extensive documentation provided in its user guide [23]. This exported file is provided to and used by the digital twin (DT) by using the SMOL interpreter. Section 3.3 further examines what a dynamic asset model entails.

RQ1 is also concerned with creating an asset model that is *extensible*. Being extensible in this context means that an operator should be able to manually add new smartphones that have no registered physical location, or add existing building infrastructure to the ontology. We provide a template for movable entities, which is used to add new movable entities to the domain according to context. More specifically, new smartphone individuals without physical location data are added to the asset model. The usage of the template is shown below. Changeable parts in the template are highlighted. This should be added to the OWL document (file), either by the server or manually by an operator:

```
### http://www.semanticweb.org/oscarlr/ontologies/2023/2/building#smartphone5
:smartphone5 rdf:type owl:NamedIndividual ,
               :MovableEntity ;
               :movableEntityId "5" .
```

3.1.2 RQ2

RQ2 asks whether we can "enable bidirectional data flow between the PT and DT, so that the digital twin gets updated sensor data from the mobile devices in the PT and sends informed decisions back".

Bidirectional data flow is data that is exchanged both ways, and not just in one direction. The DT observes the PT in near real-time and gets updated

sensor data from its physical counterpart. The DT should then send informed decisions back to it. A lot of value is derived from the DT making an informed decision on behalf of a mobile asset in the PT, and it is crucial that the appropriate smartphone user is notified.

The DT should make an informed decision if some criteria are met. For example, one can check if a person, referred to as a point, is inside a dangerous area. Using what is shown in Figure 2 as a basis, an area's two border corners are expressed as follows:

1. (x_1, y_1)
2. (x_2, y_2)

Meaning, to check if a smartphone's physical location (point) (x, y) is inside the area, we can check if the following is true

$$(x_1 \leq x \leq x_2) \wedge (y_1 \leq y \leq y_2)$$

We assume the rooms in the figure has the point (x_1, y_1) as their lower left border corner.

3.1.3 RQ3

RQ3 is concerned with how we can "separate dynamic data (smartphone's physical location) from static data (building infrastructure) in the asset model". Hypothesis (H) states that the asset model should be of a simple building. The rooms modelled as square areas constitute it. In contrast to building infrastructure, the set of mobile assets can change at any time. As described in Section 1.4, we are not concerned with other movable entities in the building domain. And, even though the data of the building infrastructure may change, it will not change during program execution in SMOL. This is not the case with the time series data accessed from InfluxDB in SMOL. One way to separate the dynamic data from the static data is by examining how often changes occur through time. Static behaviour should not be affected by dynamic behaviour. More specifically, when reloading the asset model, the building data will not be affected by the continuous changing physical locations of the smartphones. Further analysis of static and dynamic data is described in Section 3.4.1, and also later discussed in Section 6.1.

3.2 Building Domain

When considering the three research questions, we have created a possible domain including a built environment based on static data, as well as movable entities with their physical location as dynamic data, as shown in Figure 9. **Physical 2D space** is provided for the context of outside and is in two dimensions as described in Section 1.4. This means that height is not taken into

account. Note that the relationships between some entities use composition (i.e. indicates a very strong relationship between the entities, meaning that if the owning entity (e.g. **Physical 2D space**) is destroyed, so is the entity linked to it [45]) (it can be thought of as a "black hole").

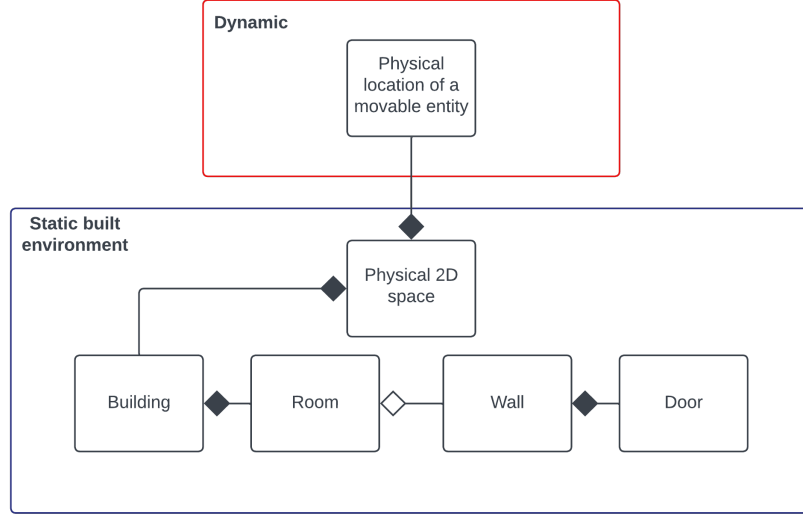


Figure 9: A UML class diagram that illustrates the separation of static and dynamic classes

3.3 Dynamic Asset Model

Section 2.3 describes concerns with BIM (Building Information Modeling), and how ontologies (asset models) created with semantic technologies deal with these concerns. Another concern with BIM worth mentioning is that it does not handle dynamic behavior very well [31].

Simply put, a dynamic asset model is an ontology in which some data changes through time. If we were to export the ontology created in Protégé, we would get an OWL document (file) that would describe the domain in terms of classes, properties, and individuals. Changing the ontology in any way (e.g. adding another movable entity) would represent a new RDF graph.

Figure 5 shows an exported ontology of a building in Turtle Syntax. The ontology editor lets us specify the format of the ontology to be exported. From creating this ontology as a basis for our future implementation, we discovered that it would be necessary to create a common class for the movable entities (individuals) due to the sheer number of them in the GUI (Graphical User Interface) in Protégé, as well as to formalize the knowledge of heterogeneous smartphones.

A digital twin (DT) based on a dynamic asset model should be able to recognize the following, according to the generated knowledge graph:

- Manual changes of static data by an operator
- Automatic updates of dynamic physical location data by the server

3.4 Data Separation

We will further analyze the separation of static and dynamic data and give some practical examples to support this. First, we describe and illustrate how they can be separated in the asset model and in SMOL. Then we examine how we can separate the static analysis of what a critical area is from the dynamic snapshots of smartphones in a certain point in time.

3.4.1 Static and Dynamic Data

Figure 10 shows the separation of static and dynamic data. The **Asset Model** do not have physical location data of the smartphones, but the smartphone individuals and their IDs as static data. The SMOL program accesses the smartphones by their IDs with **access**, and adds the dynamic data from InfluxDB according to the IDs. We take notice that a server could continuously update the time series database with the physical locations of the mobile devices, as well as add the smartphone individuals to the asset model.

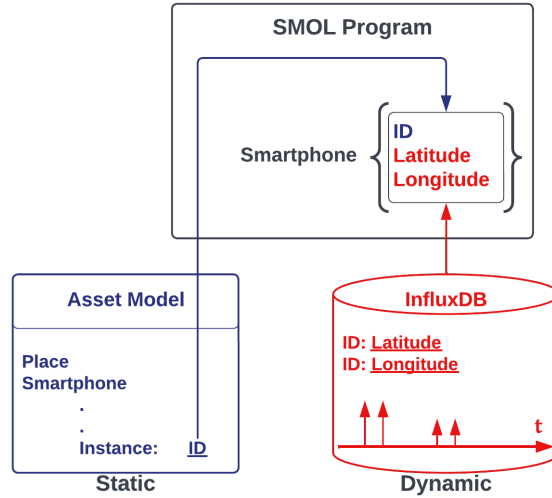


Figure 10: The separation of static and dynamic data illustrated in blue and red, respectively

The aforementioned figure also shows the direction of data flow between the **Asset Model** and **SMOL**, and **InfluxDB** and **SMOL**. The SMOL program accesses the static data (**Place** and instances of **Smartphone**) in the asset model and gets the updated sensor data of the smartphones from InfluxDB. The SMOL program then creates **Smartphone** objects from the data.

3.4.2 Further Data Separation

We take notice that when querying an asset model with a SPARQL query, the consensus is to use red color for core parts of SPARQL syntax or language, whilst the color blue is used for examples of query-specific values or text that go into SPARQL queries [54]. It is important to look at the differences between the sensor data and the informed decisions in terms of what the data can be used for. The updated sensor data is the newest physical location from the mobile assets in the PT. It is crucial that data can be used as is, and therefore the data should be exact coordinates (latitude, longitude), and identifiable. We should think about what an informed decision means. An informed decision is a conclusion made by the DT when there is sufficient information to determine something on behalf of the PT to control it. An example of this is the DT sending warnings back to the PT when the DT has determined that smartphones are inside critical areas. The analysis of what a critical area is, however, is not taken into account, as stated in Section 1.4. Therefore, the DT should not decide that an area is dangerous. This could be a job for an operator responsible for defining the areas in the asset model.

Lastly, we should separate the static analysis of what a critical area is from the dynamic snapshots of smartphones at a certain point in time. This way the dynamic behavior doesn't affect the static behavior. Some background information on a digital shadow (DS) was given in Section 2.1. A DS should be created to handle one-way data flow and to further separate data, as well as give modularity to our implementation.

3.5 Analysis of Technologies

We analyze different technologies from the descriptions of them in Chapter 2. These technologies can be used to realize the proposed high-level architecture in Figure 1. Choices of architectural components are justified based on the analysis of the problem space. We look at time series databases and real-time databases, frameworks for creating cross-platform apps, and semantic digital twins.

3.5.1 App Development

From the background information on app development provided in Section 2.6, we chose to focus on Flutter as a framework for creating cross-platform applications (apps) due to a number of reasons. We considered other frameworks for creating apps for iOS and Android from a single codebase as well, such as Ionic and React Native. However, we had no prior experience with either, and Flutter

has great documentation provided by Google, and is popular in the industry. It was also made clear in the background section on App Development that Google Maps SDK should be used to integrate Google Maps into the Flutter app.

The programming languages were seen as tools that came with the choice of frameworks and were thus not focused on that much. However, the programming language used in Flutter applications is Dart, which is also developed by Google. React Native and Ionic apps are both written in JavaScript. Ionic apps can be written using a single code base in the libraries React, Angular, or Vue [30].

3.5.2 Databases

There exist many databases, but Section 2.7 narrowed the choices down to the time series database InfluxDB and the real-time database provided by Firebase.

SMOL, as a framework for creating digital twins (DTs), supports InfluxDB and provides open-source examples for querying time series data which made it an easy choice.

When it comes to real-time databases, we saw that the real-time database by Firebase is based on web-sockets. This is more performant than polling due to data being accessed in real-time, and due to low latency. There is also no limit to the number of nodes that can be pushed to a list in Firebase, which let us handle an unlimited number of smartphones as stated in Section 1.4. Although this can be very costly if the user base scales out of proportion. Redis was also considered as a real-time database that could be used for our implementation, but we had prior experience with Firebase, and integrating it should take less time. It is a cloud-hosted solution [11], but one can download the Google Cloud SDK Shell to authenticate and interact with the services provided by Firebase.

Furthermore, using a database in the communication between clients and the server allows for an operator to easily check the assigned clients, as well as which clients should be informed. From the prototypical implementation, it was made clear that *real physical devices* should be detected by the physical twin (PT).

Lastly, we should ask ourselves why we should use a real-time database in our client-server communication, but this is later discussed in Section 6.3 where we also look at FastAPI and the communication between architectural components.

The **Server** in Figure 1 has not been explicitly described yet in this thesis as its main function was to enable a coupled flow of data between the PT and DT. Java is a well-known and established programming language, and our fellow students are quite familiar with its syntax as well. Lastly, The OWL API and Apache Jena have many open-source examples in Java, which is another reason we should use Java in our server implementation and not e.g. Kotlin (which the SMOL interpreter is written in).

3.5.3 Semantic Digital Twin

Section 2.4 describes that SMOL (Semantic Micro Object Language) can be used as a framework for creating digital twins (DTs) and that it integrates semantic

technologies. Knowledge graphs can be generated by using the **dump**-command in the REPL provided by the SMOL interpreter. The knowledge graph can capture a knowledge base, such as a dynamic asset model, in which static data is separated from dynamic data. This asset model combined with the SMOL program, produces the knowledge graph. It is also possible to access the asset model from the SMOL program. As SMOL also integrates semantic languages, such as SHACL for validation and SPARQL for querying, it could be a good fit for our implementation.

3.6 Structural Requirements

We have analyzed the research questions, the building domain, and the dynamic asset model in which static and dynamic data are separated. Technologies have also been examined and justified. From this, we create the structural requirements for the implementation, which is an *unordered* (on all levels) nested list. It is structured into the parts **Client**, **Server**, **Databases**, and **Twin**.

Structural requirements:

- **Client**
 - Can choose to send its newest status (ID, latitude, longitude)
 - Appropriate endangered clients are warned
 - Show appropriate in-app messages to the user
 - Ask for permission to track the physical location of the device
 - Clearly show when the physical location of the device is tracked
 - Clearly show when the physical location of the device is *not* tracked
- **Server**
 - **1.** Forward the sensor data to a time series database
 - **2.** Forward the informed decisions to a real-time database
 - Should be possible to do **1.** and **2.** above at the same time
 - Reload asset model with tracked clients
 - Access and read the generated knowledge graph
- **Databases**
 - Realtime database
 - * Store the ID and physical location of the clients
 - * Store data about which clients the DT made an informed decision for
 - Time series database
 - * Store the time series data

- **Twin**

- **3.** DS gets the newest sensor data from the time series database
- **4.** Access the smartphones in the asset model by their IDs
- Create smartphone objects by combining data from **3.** and **4.** above
- Access the critical areas from the asset model
- Check if smartphones are inside any critical areas
- Handle new smartphones that don't have a physical location
- Provide examples of SPARQL queries
- Show informative messages in the REPL during program execution

3.7 Design

Having the structural requirements described in Section 3.6 for the clients in mind, in addition to the analysis of app development. And from the background information on GPS sensors in Section 2.6, a wireframe was created. We justify creating this without any input from conducting user studies because it can be seen as a standard map screen in an app, due to similarities with the official one. We should however create our own version of the location button, referencing the requirements of the client. Figure 11 shows the wireframe of the User interface (UI) of the map screen in the app. Simply put, the UI is what the user sees in the app, and is built from the widgets in Flutter, behaving much like components in e.g. React.

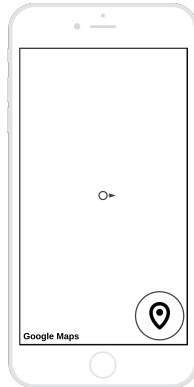


Figure 11: Wireframe of the UI in the app

4 Implementation

This chapter describes the publicly available proof-of-concept implementation⁹, based on the structural requirements in Section 3.6, by describing each of the following parts in it:

- Client
- Server
- Databases
- Twin

In the first section, we describe the implementation of a prototype based on the initial architectural components in Figure 1.

Then we look at the final version in Figure 12, and describe what was implemented. For each of the larger architectural components in the figure, we reference the section that describes it in Chapter 2, and the design decision that enables it in Chapter 3. Along the way, we discuss challenges when developing, as well as which requirements were met and why. Due to this, the last section presents the overall evaluation of the implementation in terms of the structural requirements. Further evaluation is presented in Chapter 5.

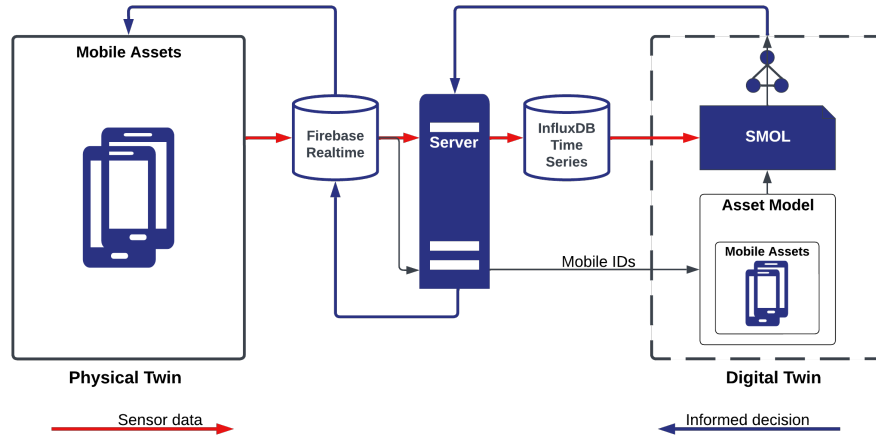


Figure 12: Overview of architectural components in the final implementation

4.1 Prototypical Implementation

Due to the extensive architecture, we chose to implement a prototype by letting some of the architectural components in Figure 1 communicate first. The

⁹<https://github.com/Owlar/Master-Thesis>

Physical Twin in the figure consists of **Mobile Assets** which are smart-phones. Based on the background information in Section 2.6, and from further narrowing of the technologies in Section 3.5.1, a Flutter app was created for cross-platform support.

We used the wireframe in Figure 11 in the app prototype. Figure 13 and Figure 14 show the location recording toggle function off and on, respectively. The recording is off by default, and when the button is pressed, it starts tracking the physical location of the device and changes color. The latest physical location of the device is sent, alongside its ID, as a status message to the server over TCP (Transmission Control Protocol).

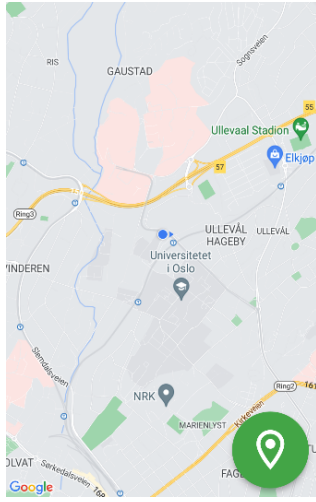


Figure 13: Physical location recording is *off* (green button)

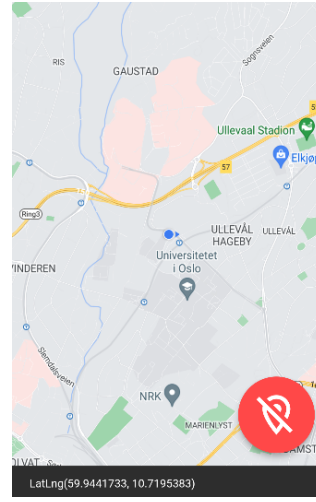


Figure 14: Physical location recording is *on* (red button)

We created a Maven project (a tool for managing the project) and used Java for the server implementation. To let the server track many clients, we first assigned an ID to the client when it connected to the server which kept track of joining clients. The client received its assigned ID from the server which was later sent alongside the client's physical location as status update messages. As this was an early adaptation, we implemented client-server communication over TCP (Transmission Control Protocol).

An asset model of a simple building was also created, as shown in Figure 2 (this time with coordinates as data properties which were "added" to individuals of the `MovableEntity` class suggested in Section 3.3. We used Protégé to create the domain and exported it as an OWL document (file). This file was then accessed in a new SMOL program based on the background information in Section 2.2.1, Section 2.4, and from further analysis in Section 3.4.1.

To sum up, in the early version of the implementation a simple app was

created to send sensor data (physical location from GPS-sensor in the devices as described in Section 2.6) to a server.

4.2 Implemented Solution

Figure 12 shows the architectural components in the final implementation. More specifically, it illustrates the **Mobile Assets** in the **Physical Twin**, its **Digital Twin** based on a dynamic **Asset Model**, and the **Server** which enables full communication between the PT and DT both ways. In addition to this, the figure illustrates that the database **Firebase Realtime** acts as a channel between the **Physical Twin** and the **Server**, and that **InfluxDB Time Series** is used for storing the sensor data which the SMOL program accesses, as shown in the example function in Figure 8.

In the following sections, we describe the architectural components in the figure based on the background information and the analysis, and by referencing the corresponding *part* (Client, Server, Databases, or Twin) in the structural requirements.

For further reference on how we set up SMOL to be used in this thesis, see Section 5.1.

4.2.1 Client

A Flutter app was written in Dart, based on the background information of app development in industry in Section 2.6, and from the analysis of related technologies and their benefits in Section 3.5.1. From the paragraph about GDPR (General Data Protection Regulation) in Section 2.6 and the requirement "Ask for permission to track the physical location of the device" we implemented location permissions. Section 6.5 discusses this further. The map screen in the app was created with the wireframe shown in Figure 11. We did not implement a login screen because of the context of a crisis situation, and due to the limited data stored in Firebase.

The source code of the app was structured into the following parts:

- `main.dart` which initializes the app and Firebase
- `map.dart` which contains code for the UI (User interface) in the app. This is done by rendering Flutter widgets.
- `service.dart` accesses Firebase, and writes to, or reads from it.
- the model-class `status.dart` is used to create status objects which are sent as JSON to Firebase in which it is stored as a JSON object, as described in Section 2.7.

We used the Google Maps SDK (Software Development Kit) in the implementation to integrate the map and its functions in the app, as described in Section 2.6.

There are two requirements stating that we have to clearly show when the recording function is off/on. In order to clearly show it for those that are color blind as well, the colors of the button changed to blue and red, respectively. Figure 15 shows the final color theme, which is located in the previously mentioned file `main.dart` and can be accessed from anywhere in the app.

```
final ThemeData _theme = ThemeData(
  colorScheme: ColorScheme.fromSwatch().copyWith(
    primary: const Color(0xFF2A3282),
    secondary: const Color(0xFFE81313),
  )
);
```

Figure 15: The color theme in the app

The Dart code in Figure 16 shows the conditional rendering of the location recording button which uses the color theme above. Since the Flutter widgets are rendered and the color of the button was initially set, we had to keep track of the state to change the color of the button. This was done by adding new status messages to `_messages` or emptying the list, in `setState(() { });`. The underscore as the leading character in `_messages` means that the member is only visible in its class, much like using `private` for members in Java. The code also shows the use of a ternary operator, which is syntactic sugar for the conditional operators `if` and `else`.

```
floatingActionButton: _messages.isEmpty ? FloatingActionButton.large (
  onPressed: () => _start(),
  backgroundColor: Theme.of(context).colorScheme.primary,
  child: const Icon(Icons.location_on_outlined, size: 60),
) : FloatingActionButton.large (
  onPressed: () => _stop(),
  backgroundColor: Theme.of(context).colorScheme.secondary,
  child: const Icon(Icons.location_off_outlined, size: 60)
),
```

Figure 16: Rendering of the location recording button in the app

Furthermore, when recording of the physical location stops, the appropriate (endangered) clients are warned, as required in in the structured requirements. We made sure the user could close and re-open the app, and still be warned when stopping location recording. However, re-opening the app after the process is destroyed assigns a new ID.

Firebase stores the informed decision, which the server forwards from the twin. Figure 17 and Figure 18 show the conclusion of being *safe* and *endangered*, respectively.

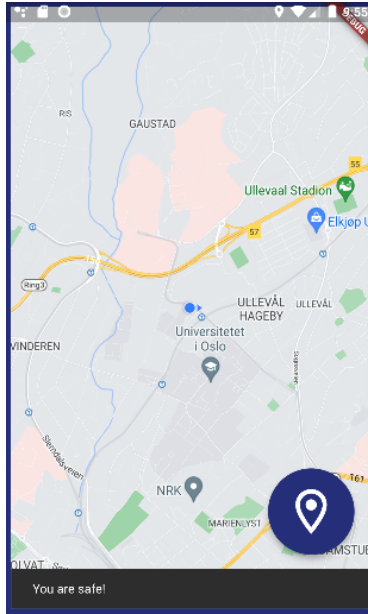


Figure 17: The mobile asset (smartphone) in the PT is *safe*, as concluded by the DT

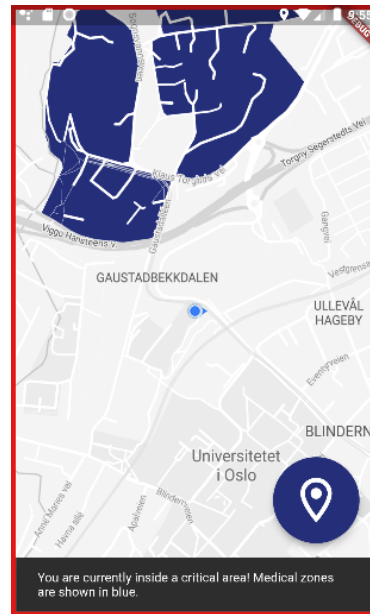


Figure 18: *Endangered* smartphone from the informed decision to be inside a critical area

The status message is sent by using the reference to the Firebase document, as shown below.

```
Future<void> addStatus(Status status) async {
    DatabaseReference reference = FirebaseDatabase.instance.ref("mobiles/${status.id}");
    await reference.set({
        "id": status.id,
        "latitude": status.latitude,
        "longitude": status.longitude
    }).onError((error, stackTrace) => null);
}
```

Figure 19: Sending its status to Firebase

After the status is sent, streaming of the physical location resumes. We also added a distance filter to prevent unnecessary writing.

4.2.2 Server

The server enables full communication both ways simultaneously between the PT and the DT. Sensor data from the PT are forwarded to the DT through InfluxDB, whilst informed decisions are forwarded to Firebase. This is done

```

public void getSensorData() {
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                String json = new
                    ↳ Gson().toJson(snapshot.getValue(Object.class));
                Data data = new Gson().fromJson(json, Data.class);
                influxDB.insertDataPoint(data);
                System.out.println("Forwarded sensor data to DT!");
                try {
                    Owl.addIndividual(data);
                } catch (OWLOntologyCreationIOException e) {
                    e.printStackTrace();
                }
                hasSensorData = true;
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
            System.out.println("Could not retrieve data from realtime
                ↳ database.");
        }
    });
}

```

Figure 20: Get updated sensor data from Firebase

simultaneously in intervals by using threads. Thus the server enables seamless bidirectional data flow between the outermost architectural components in Figure 12, referencing the requirements for the server in Section 3.6. When the server runs, connections to Firebase and InfluxDB (time-series database) are first established. Then the server forwards sensor data to the DT by sending the updated status of clients in the PT to the time-series database, assuming there is sensor data. The smartphones are also added to the asset model with their assigned IDs. Figure 20 shows the implemented solution to get updated sensor data from Firebase. Note that we had to implement a workaround (from Gson to Gson) because of the `@Measurement`-annotation in the `Data`-class, in order to create objects of it.

Figure 21 has a call to a function in the `Owl`-class. The line `Owl.addIndividual(data);` calls the function with the only parameter being the specific timestamped status message. In the function being called, we ensured the static data in the file (building.owl) written to was not affected by the data.

Figure 21 shows how the informed decision from the DT is forwarded to Firebase.

```

private void warnEndangeredClients(Map<String, String> results) {
    reference =
        ↪ FirebaseDatabase.getInstance().getReference("endangered");
    DatabaseReference.CompletionListener completionListener =
        ↪ (databaseError, databaseReference) -> {
        System.out.println("Forwarded informed decision to PT!");
        hasWarned = true;
    };
    reference.setValue(results, completionListener);
}

```

Figure 21: Forward informed decision to PT through Firebase

The server prepares to send sensor data to the time-series database whenever there are changed clients in the PT, such as an updated physical location, by keeping a reference to the list. Whenever a change occurs, the server forwards the sensor data of all tracked clients. The DT gets the overall picture of the different locations at a specific point in time. Figure 22 shows how a measurement point is inserted into InfluxDB. As can be seen in the figure, the programming language used is Java. A point is made from the model-class `Data`. More specifically, the instance of the `Data`-class is a timestamped status message.

```

public void insertDataPoint(Object object) {
    WriteApiBlocking writeApi = db.getWriteApiBlocking();
    writeApi.writeMeasurement(bucket, org, WritePrecision.MS, object);
}

```

Figure 22: Inserting a measurement point (timestamped status message) into the Data-bucket in InfluxDB

According to the structural requirements, the server should reload the asset model and access the generated knowledge graph from the DT. We put the exported OWL document (file) into the file structure of the DT (we later discuss how data can be stored, the reasons for using files, and the limitations to this in Section 6.2. The server writes by using the Java framework OWL API which is described in Section 2.5. Based on the background information of semantic technologies in Section 2.2.1, and the example of adding **individuals** of **MovableEntity** to the asset model (ontology) in Section 3.4.1, we created a function for adding data of a smartphone to the file. Before using OWL API, we tried using Apache Jena to reload the asset model, but it could not write data in Turtle syntax to the asset model. We suspect some compatibility issues with the Turtle syntax in the file exported from Protégé.

Apache Jena was used to access the endangered mobile assets in the knowledge graph which was generated from a combination of the knowledge base and the state of the SMOL program in the DT. More specifically, a model of the

RDF graph was loaded, and the informed decision was forwarded to Firebase to "close the loop" of the bidirectional flow of data between the PT and the DT, as shown in Figure 12.

4.2.3 Databases

We shortly describe how data is stored in Firebase and InfluxDB. In Firebase, the assigned clients are stored as unstructured nodes in the list `mobiles`. As discovered in the analysis of technologies in Section 3.5.2, the number of nodes in a list are limitless. The IDs of the endangered smartphones are also stored there, and accessed by a function in `service.dart` in the app. Time-series data is stored in InfluxDB in a bucket named `Data`, in which there is sensor data that the SMOL program in the DT accesses. From this, the structural requirements regarding databases are met.

4.2.4 Twin

From the background information of DTs in Section 2.1 we created a digital twin, based on a dynamic asset model of a simple building, that handled mobile assets. The asset model had areas with predefined coordinates (latitude1, latitude2, longitude1, and longitude2) The DT and its parts were semantically enriched by using the semantic technologies presented in Section 2.2.1, as well as by using SMOL as an enabler of semantics from the analysis of it in Section 3.5.3.

The semantic digital twin was created in SMOL by using its core language illustrated in Section 2.4. And by using the SMOL Interpreter (`smol.jar`) in the file structure to read and execute the SMOL program. The SMOL program consists of two modular parts, namely:

1. **DS** (Digital Shadow) has two purposes. First, it acts as a timestamped snapshot of smartphones at a certain point in time. From the background information in Section 2.1.2, we abstract this so that the asset is all tracked smartphones. Secondly, it is used to access the newest sensor data, from the descriptions in Section 3.4.2. It accesses the latest physical locations from a specified time ago and only gets sensor data. This is based on the descriptions in Section 2.7, and from the example in Figure 8. This can be seen in the function in Figure 23. Note that it looks like we get outdated sensor data (**first** added to the bucket), but in fact we get the latest. Also note that when this was implemented there was no support for single-quote string delimiters in **access**.
2. **Digital Twin** is a digital representation of the PT. Its SMOL program accesses:
 - (a) the critical areas in a building domain defined in the asset model
 - (b) the smartphones by their IDs from the asset model
 - (c) the updated physical location of the smartphones from the asset model, based on a revised version of Figure 8.


```

List<Double> getSmartphoneCoordinates()
  List<Double> coordinates = null;
  coordinates = access(
    "from(bucket: \"Data\")
      |> range(start: -20m)
      |> filter(fn: (r) => r[\"_measurement\"] == \"data\")
      |> filter(fn: (r) => r[\"_field\"] == \"latitude\" or
        ↪ r[\"_field\"] == \"longitude\")
      |> sort(columns: [\"_time\"], desc: true)
      |> unique(column: \"id\")
      |> aggregateWindow(every: 5m, fn: first, createEmpty:
        ↪ false)
      |> yield(name: \"first\")",
    INFLUXDB("influx.yml")
  );
  return coordinates;
end

```

Figure 23: Accessing the updated sensor data from InfluxDB

From this, smartphone objects are created by combining the data as shown in Figure 24 by matching the IDs accessed from InfluxDB and the smartphones in the asset model. The *first four requirements* in the twin-part in the structural requirements in Section 3.6 are thus met.

```
MovableEntity smartphone = new MovableEntity(id, lat, long, False);
```

Figure 24: Creating a smartphone object from the smartphone ID in the asset model, in combination with the corresponding updated coordinates from InfluxDB

As described in Section 2.3, a DT should allow for interoperability between various sources of data. By using the common class `MovableEntity` for the set of movable heterogeneous entities, interoperability is gained in the DT.

Referencing the requirement "Check if smartphones are inside any critical areas", an example is described in Section 3.1.2, where an informed decision can be made if the criteria evaluate to be true, assuming sufficient data. The code implementation of it is shown in Figure 25, where the function is located inside the `Area`-class. Meaning, we write `area.containsPoint(lat, long)` to check if `area` contains the smartphone with latitude (`lat`) and longitude (`long`).

```

Boolean containsPoint(Double lat, Double long)
    Boolean inLatitudeRange = (lat >= this.latitude1) & (lat <=
        ↪ this.latitude2);
    Boolean inLongitudeRange = (long >= this.longitude1) & (long <=
        ↪ this.longitude2);

    if (inLatitudeRange & inLongitudeRange) then
        return True;
    end

    return False;
end

```

Figure 25: Checking if a smartphone is inside an area programmatically

If the DT concludes that a smartphone is inside *any* of the areas (set to critical by an operator) which are accessed from the domain, the twin sets `smartphone.endangered = True`; and adds the object to the list of endangered smartphones for the specific area, which is shown in Figure 26. Meaning, we can easily query the endangered smartphones using the critical area. Note that what is shown in the figure is inside a while-loop, hence the use of the counter `j`.

```

a = areas.get(j);
isInsideACriticalArea = a.containsPoint(lat, long);
if (isInsideACriticalArea) then
    smartphone.endangered = True;
    a.smartphones = new List<MovableEntity>(smartphone, a.smartphones);
end

```

Figure 26: Checking if a smartphone is inside an area that is set to critical. If the area contains its point (x, y), the smartphone is set to endangered, and is added to the list for the specific area

Regarding the last requirements of this part, informational messages, such as examples of SPARQL queries, are shown in the REPL during program execution. Figure 27 show how we can *manually* check which smartphones are inside a critical area, and thus are endangered, and the results (information about endangered smartphones). The querying is done in the SMOL REPL, which is started from the information in Section 2.4.1.

```
MO> query SELECT ?id ?endangered ?obj WHERE{ ?obj prog:MovableEntity_endangered true. ?obj
prog:MovableEntity_movableEntityId ?id. ?obj prog:MovableEntity_endangered ?endangered }
MO-out>
```

id	endangered	obj
"4"	true	<https://github.com/Edkamb/SemanticObjects/Run1683913332038#obj55>
"3"	true	<https://github.com/Edkamb/SemanticObjects/Run1683913332038#obj57>
"2"	true	<https://github.com/Edkamb/SemanticObjects/Run1683913332038#obj59>
"1"	true	<https://github.com/Edkamb/SemanticObjects/Run1683913332038#obj61>

```
MO>
```

Figure 27: Querying with SPARQL

Although the DT knows of manually added smartphones the system cannot further track them. This means the requirement "Handle new smartphones that don't have a physical location" is not *fully* met.

Lastly, Section 3.5.3 describes the use of `dump` in the REPL (Read-eval-print loop) to generate the knowledge graph, which the server uses by doing the following:

1. Creates a model of the knowledge graph with Apache Jena
2. Reads the model
3. Iterates over the resources (smartphones) in it
 - If the smartphone is endangered, the ID of it is added to a list
4. Adds the list of IDs to Firebase

Clients in the PT then access Firebase by toggling the button, and are warned if they are in danger from the informed decision, which closes the loop.

4.3 Implemented Requirements

This section *presents* the overall evaluation of the developed implementation in terms of which structural requirements in Section 3.6 are met, based on the analysis in Chapter 3. This is shown in Table 2. The Structural requirements were divided into four parts, one for each of the *main* architectural components shown in Figure 12.

	Requirement	Status
Client	Can choose to send its newest status (ID, latitude, longitude)	✓
	Appropriate endangered clients are warned	✓
	Show appropriate in-app messages to the user	✓
	Ask for permission to track the physical location of the device	✓
	Clearly show when the physical location of the device is tracked	✓
	Clearly show when the physical location of the device is <i>not</i> tracked	✓
Server	1. Forward the sensor data to a time series database	✓
	2. Forward the informed decisions to a real-time database	✓
	Should be possible to do 1. and 2. above at the same time	✓
	Reload asset model with tracked clients	✓
	Access and read the generated knowledge graph	✓
Databases	Firebase: Store the ID and physical location of the clients	✓
	Store data about which clients the DT made an informed decision for	✓
	InfluxDB: Store the time series data	✓
Twin	3. DS gets the newest sensor data from the time series database	✓
	4. Access the smartphones in the asset model by their IDs	✓
	Create smartphone objects by combining data from 3. and 4. above	✓
	Access the critical areas from the asset model	✓
	Check if smartphones are inside any critical areas	✓
	Handle new smartphones that don't have a physical location	×
	Provide examples of SPARQL queries	✓
	Show informative messages in the REPL during program execution	✓

Table 2: Which structural requirements were met

The only requirement that was not fully met, was "Handle new smartphones that don't have a physical location". The SMOL program accesses the asset model and creates objects from all smartphones in it, but only adds the physical location to the smartphones that have their data in InfluxDB. There is no further support for adding the latitude and longitude to the objects. There is error handling in the SMOL program, and the REPL prints information regarding this. We also assume that an operator *knows* how to add smartphones to the asset model by either loading it into an ontology editor or use the template described in Section 3.1.1.

5 Evaluation

This chapter evaluates the developed implementation from various experiments. They are performed in a real-life environment; on a public network with physical devices. We start by shortly describing how we set up SMOL to be used in the implementation. Then we provide guided test runs of the server automatically reloading the asset model, and of an operator manually editing the asset model by adding smartphones to it. We *demonstrate* the evaluation of requirements in Table 2.

Then we evaluate how long it takes to manually warn endangered clients. We also evaluate whether the system is correctly twinned based on detected smartphones, and plot the results in the Data Explorer view in InfluxDB. This was a real experiment conducted at IFI.

5.1 Setup of SMOL

To set up SMOL in the implementation in this thesis, we cloned the GitHub repository¹⁰ with:

```
git clone https://github.com/smolang/SemanticObjects.git
```

Then we created jar-file named `smol.jar` from the source code. It was put in the created folder at the relative file path `master/twin`. Then we navigated into it with `cd master/twin` and used the SMOL interpreter (`smol.jar`) as described in Section 2.4.1.

5.2 Guided Test Runs

This section provides guided test runs of the implemented solution. We describe what is going on, illustrate the running components, and examine it in terms of the requirements.

Before we start the test runs, we empty the databases, remove smartphones from the asset model, run the SMOL program, and dump to the knowledge graph. To set it up inside the `twin`-directory, we need the file `smol.jar`. In the terminal we use:

```
java -jar smol.jar -d http://www.semanticweb.org/oscarlr/ontologies/2023/2/building#  
-b building.owl
```

From this, the REPL was ready to use, and the knowledge graph reflected that there were no tracked smartphones.

5.2.1 Automatically Reloading Asset Model

As the scene was set, we first start the server. Then we run the SMOL program in the DT, and as expected, there were no time-series data, nor any detected clients, as no smartphones had sent their physical location yet.

¹⁰<https://github.com/smolang/SemanticObjects>

We press the button in the app to send the updated status. The server detected it, as shown in Figure 28. It keeps a reference to the list of status messages and forwards all of it whenever there is a change.

```
Server has started! Getting clients from realtime database.
Setting up real time database..
Setting up time series database..
Running..
Forwarded sensor data of ID: 1 to DT!
```

Figure 28: Server detects a smartphone

The status messages are timestamped before writing to InfluxDB. When the DS in the twin gets this data, the DT has a picture of the smartphones at a certain point in time. Figure 29 shows the time-series data in the graph view in InfluxDB. We can see that the sensor data is constantly forwarded. Figure 10 partly illustrates this.

Based on the background information in Section 2.3, the server adds the smartphone individual to the asset model according to the template described in Section 3.1.1.



Figure 29: Time-series data in InfluxDB

We add two new heterogeneous smartphones to the pool of tracked clients, and we check the different physical locations in the map view in the time-series database, which is illustrated in Figure 30. In the figure, different clients have different colors. We used the `shapeData()`-function to rename the coordinates and get a unique ID, namely `s2_cell_id`. Colorized thresholds were added to differentiate based on ID. The figure also shows the following code:

```
import "experimental/geo"

from(bucket: "Data")
```

```

|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "data")
|> filter(fn: (r) => r["_field"] == "latitude" or r["_field"] == "longitude")
|> geo.shapeData(latField: "latitude", lonField: "longitude", level: 1)
|> map(fn: (r) => ({r with _value: r.id, _field: "id"}))
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)

```

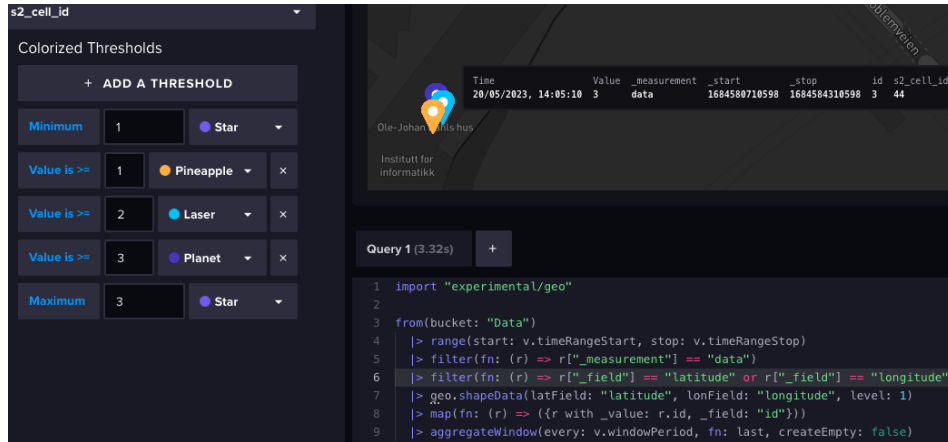


Figure 30: Time-series data represented in a map in InfluxDB from using a Flux query

After this we run the SMOL program with `reada main.smol`, and the DT concluded that all of the clients were inside the critical area (IFI). We can also see that **4. All smartphone in domain have a physical location** shown in Figure 32. We run a SPARQL query in the REPL to verify that they are endangered.

```

query SELECT ?id ?endangered ?obj WHERE{
  ?obj prog:MovableEntity_movableEntityId ?id.
  ?obj prog:MovableEntity_endangered ?endangered
}

```

We can also evaluate that static data is not affected by the reloading of dynamic data. Static data was queried with SPARQL before and after the assets were added, and compared. Git was also used to confirm it. This can be done from effectively seeing it in a GUI, such as GitHub Desktop, or using `git diff building.owl`, as shown in Figure 31.

```
(base) oscarlr@eduroam-193-157-192-214 twin % git diff building.owl
diff --git a/twin/building.owl b/twin/building.owl
index 2e46a8f..1d80ae0 100644
--- a/twin/building.owl
+++ b/twin/building.owl
@@ -148,4 +148,10 @@
     :name "Big Critical Area" .

+### http://www.semanticweb.org/oscarlr/ontologies/2023/2/building#smartphone1
+:smartphone1 rdf:type owl:NamedIndividual ,
+              :MovableEntity ;
+              :movableEntityId "1" .
+
+### Generated by the OWL API (version 5.1.20.2022-02-19T16:38:31Z) https://github.com/owlcs/owlapi/
(base) oscarlr@eduroam-193-157-192-214 twin %
```

Figure 31: Smartphone individual with ID 1 is the only change in the file

We then generate the knowledge graph which the server reads from. The informed decision is forwarded to the PT. The result from this was that the appropriate clients were warned in the app when they pressed the button again, as shown in Figure 18. The test run was considered a success, and it confirms the coupled flow of data shown in Figure 12. This run can be done again by simply pressing the button in the app again.

```
4. All smartphones in domain have a physical location
---
5. Printing all smartphones as instances of prog:MovableEntity:
1
59.9435374
10.7183385
True
2
59.9435641
10.7183953
True
3
59.9435854
10.7183582
True
---
6. Smartphones with known physical locations that can be checked:
1
2
3
query SELECT ?id ?endangered ?obj WHERE{ ?obj prog:MovableEntity_movableEntityId ?id. ?obj prog:MovableEntity_endangered ?endangered }
MO> query SELECT ?id ?endangered ?obj WHERE{ ?obj prog:MovableEntity_movableEntityId ?id. ?obj prog:MovableEntity_endangered ?endangered }
MO-out>

=====
| id | endangered | obj |
=====
| "3" | true | <https://github.com/Edkamb/SemanticObjects/Run1684584461934#obj47> |
| "2" | true | <https://github.com/Edkamb/SemanticObjects/Run1684584461934#obj52> |
| "1" | true | <https://github.com/Edkamb/SemanticObjects/Run1684584461934#obj57> |
=====
MO>
```

Figure 32: Informative messages in the REPL from running the SMOL program in the DT

As a last test, since the smartphones were already added to the asset model, but had updated locations, we ran the server again, then the SMOL program again (without exiting the REPL), and got the updated locations. Meaning, when the set of IDs remains the same, the DT reflects its physical counterpart much faster. Nonetheless, it is naive to assume only the physical locations change.

5.2.2 Manually Adding Smartphones

We will evaluate manually add smartphones to the asset model by using the previous test run as a basis. In the ideal scenario, taking into account what is stated in Section 4.3, it should be known which smartphones have no physical location. We use the description of manually adding a new smartphone at the end of Section 3.4.1.

We **exit** the REPL, load the asset model again, and execute the program. The result is that the message "**4. All smartphones in domain have a physical location**" in Figure 32 is *replaced* with what is shown in Figure 33. The figure also shows more informative messages in the SPARQL endpoint. The knowledge graph successfully reflected that it was *known* that a smartphone with ID 4 had no physical location. Although it can't be tracked, the test run was considered a success.

```
4. Printing the mobile assets without a physical location
4
null
null
null
null
-----
5. Printing all smartphones as instances of prog:MovableEntity:
1
59.9435374
10.7183385
True
2
59.9435641
10.7183953
True
3
59.9435854
10.7183582
True
4
null
null
null
-----
6. Smartphones with known physical locations that can be checked:
1
2
3
query SELECT ?id ?endangered ?obj WHERE{ ?obj prog:MovableEntity_movableEntityId ?id. ?obj prog:MovableEntity_endangered ?endangered }
MO>
```

Figure 33: Printing any smartphones without a physical location

5.3 Manually Warning

5.3.1 Goal

Although not the main focus of this thesis, as we are to formalize knowledge, it is nonetheless interesting to take a look at performance. More specifically we are presenting how long it takes to warn an endangered smartphone, concluded by the DT.

5.3.2 Experimental Setup

The experiment will be conducted in a manual manner. More specifically, we start components, use commands, toggle the button in the app on and off, and wait for the knowledge graph to be generated. Before each run, The REPL is exited, and the asset model is loaded into SMOL. We will manually use commands, first to load the asset model, and then to read and execute (`reada main.smol`) the SMOL program at the same time. We also type `auto`, and dump to the knowledge graph in each run. Then we wait for the server to forward the informed decision.

This evaluation as well as the previous test runs were performed on a MacBook Air with an M1 (2020) chip with 8 cores, and 8GB of RAM.

Again we test with physical devices over a public network as described in Section 5.2. The elapsed time is evaluated by subtracting the end time (stops recording physical location) from the start time (starts recording physical location).

5.3.3 Discussion of Results

Table 3 shows the elapsed time of the runs listed in ascending order by time.

Run	Elapsed time in seconds
1	15.304
2	17.143
3	17.197
4	17.447
5	17.569
6	17.732
7	18.847
8	18.939
9	19.574
Median time	17.569

Table 3: Median time of manually warning clients

The results are presented as the *median* of nine runs, because the elapsed times fluctuated. This is because the server reads the knowledge graph in intervals and because it takes some time to generate the knowledge graph. An ideal

approach in terms of performance at the expense of ensuring formalized knowledge would have been to read from the asset model directly from the server. However, passing through a critical area is less dangerous than occupying it for some time. The DT has to conclude that some smartphones are inside a critical area and set them to endangered, and writing to the asset model from the SMOL program is currently not supported. This means that the time it takes to warn clients is mainly limited by the time it takes to generate the knowledge graph. This is further discussed in Section 6.6. The appropriate client was warned, as expected from the way IDs are assigned. This was the case even when closing and re-opening the app. We conclude that formalized knowledge should be ensured, and not at the expense of possibly a false positive.

5.4 Many smartphones

5.4.1 Goal

We evaluate in terms of the number of smartphones in the domain, by *physically walking around at IFI toggling the recording button on and off*. We will look at if the server forwards all sensor data, but also check that the DT is a digital representation of the PT. The DT should conclude which smartphones are endangered. We are not concerned with if we can accurately check, as described in Section 1.4, and we assume an operator has put sufficient area data. Since we don't have access to many heterogeneous smartphones in the literal sense, we will instead conduct the experiment with one physical device running the app. This means we can not test for heterogeneity, although this is done in Section 5.2.1 anyways.

5.4.2 Experimental Setup

We will edit the source code a little to ensure a new ID is assigned every time the physical location is sent from the smartphone. The problem with this, however, is the trade-off in terms of performance compared to the ideal scenario. By using different physical devices the performance would be better since we are not constantly sending the updated physical location to Firebase and re-rendering the UI in one app. Taking all this into account, when the server gets the sensor data, there will not be much of a difference between this scenario and an ideal scenario. After we have gathered the initial data, we will start the server, to see if all sensor data is forwarded, and then detected in the DT.

5.4.3 Discussion of Results

234 unique nodes were added to Firebase, and then written to the asset model when the server was started. The SMOL program executed in almost no time. Writing to the asset model did not take much time at all.

Figure 34 shows the physical locations in the map view. Since we sent from one physical device this time, the pins have the same color with the only exception of the pin with ID 234 which has the color Ruby (although hard to

see when we are "Thunder struck"). The information window in the figure is shown by hovering over this specific point.

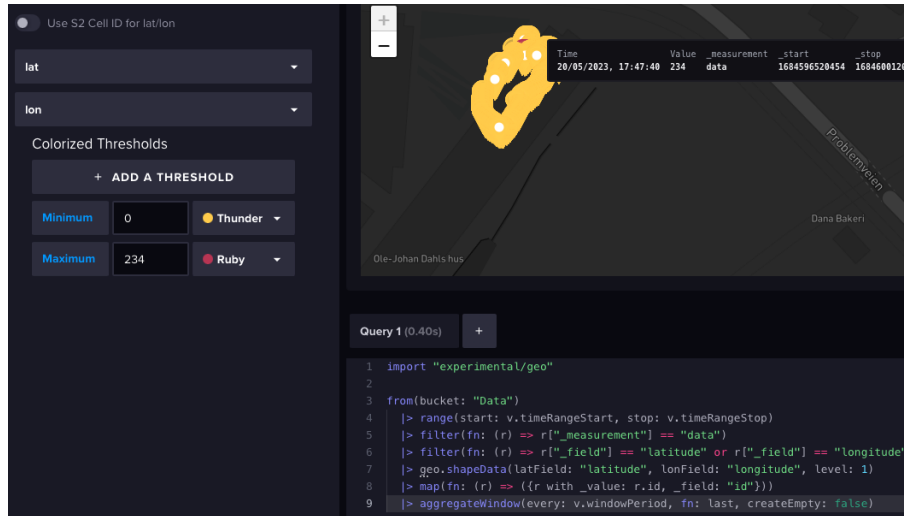


Figure 34: Map showing 234 smartphones

We will now do various simple checks with SPARQL queries in the SMOL REPL. The informative messages in the implemented SPARQL endpoint was used. First, we check how many smartphones are concluded by the DT to be inside the critical area, and are thus *endangered*. Figure 35 shows that the following SPARQL query was used:

```
query SELECT (Count(?id) AS ?IDs) WHERE{
  ?obj prog:MovableEntity_endangered ?id.
  ?obj prog:MovableEntity_endangered true
}
```

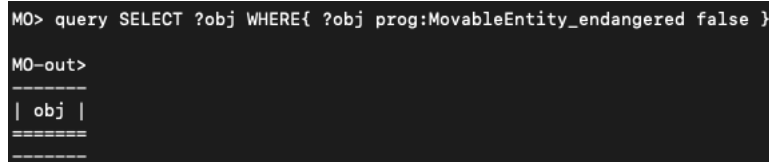
```
MO> query SELECT (Count(?id) AS ?IDs) WHERE{ ?obj prog:MovableEntity_en
dangered ?id. ?obj prog:MovableEntity_endangered true }
MO-out>
=====
| IDs |
=====
| 234 |
=====
```

Figure 35: All smartphones are endangered

In addition, we check if any of the 234 smartphones are not endangered, as follows:

```
query SELECT ?obj WHERE{
  ?obj prog:MovableEntity_endangered false
}
```

The result is no objects, hence all are endangered, as shown in Figure 36.



```
MO> query SELECT ?obj WHERE{ ?obj prog:MovableEntity_endangered false }
MO-out>
| obj |
=====
```

Figure 36: None are safe, which means all are endangered

This resulted in all 234 smartphones being endangered. From this, the DT was evaluated to be a near-real time digital representation of the PT. However, it took a very long time to dump the knowledge graph and formalize the knowledge. The number of lines written to the file was approximately 14.000. This means it will take some time for the PT to mirror the DT back. The source code and results are documented in a commit¹¹.

¹¹<https://github.com/Owlar/Master-Thesis/commit/1b1e5150593c8fc248b793b8b52845d053ee2894>

6 Discussion

In this chapter, we look at the discoveries in this thesis and compare them to other existing theories and results. Along the way, we discuss limitations with what was implemented and alternatives that could deal with them. In the last section, we discuss the implementation more in-depth.

6.1 Dynamic Data

In this thesis, we presented a way to separate data, mainly the two extremes: static data of building infrastructure, and the physical location of smartphones as dynamic data. We looked at how often data changes through time. Confirming a single change is shown in Figure 31. However, this depends on the server being able to add the sensor data to InfluxDB fast enough to differentiate it from other data that may be considered static, but also because data of static building infrastructure is only static if it is not modified. From this, an alternative is presented to illustrate more shades of blue and red in Figure 37. In this figure, we see that we *can* put **Building Infrastructure** as more static compared to **Physical Location**, but we don't *have* to. It depends on the domain, described in Section 3.2 Also meaning, the ordering of the documents above the arrow in the figure can be reordered in any way, and removed for that matter.

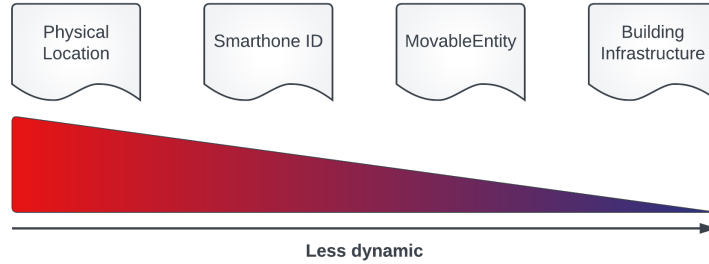


Figure 37: Shows to what extent data can be considered dynamic

6.2 Data in the Twin

In the digital twin, we are storing data in files. The asset model is stored as the file (`building.owl`), and the knowledge graph is stored in a file named (`output.ttl`). The proposals from Waszak et al. and Gulnes provide solutions for storing data in graph databases, such as Neo4j. This allows for better structuring of the data, as well as better representations of the graphs. Due to this, there are limitations with the implementation in this thesis as we store RDF graphs in files. Storing and exporting RDF graphs (such as the data in

the asset model) to a graph database like Neo4j would be ideal. Instead of updating a graph database with the updated knowledge graph, produced from the combination of the program state and knowledge base, it could be considered feasible to store it in the twin. However, this means the server must know where the file is located in the local structure, as well as be able to access it. A better solution would therefore be to store it in a graph database and then get this data over the internet, assuming we want a clear architectural separation of the server and the twin. The architecture in this thesis is already very extensive and therefore we had to make some choices on what to focus on.

6.3 Client-server Communication

We looked at web sockets and FastAPI¹² to decide how the clients should communicate with the server. Although another solution [62] uses FastAPI in architectural interconnection (not between client and server in this case), Firebase real-time database was chosen as a channel. This was because of previous experience with integrating it with the cross-platform framework Flutter, and due to intuitively being able to modify data there. The difference between Firebase and FastAPI in terms of performance remains unclear, however, but as Firebase is based on web sockets, its use was justified. Lastly, the choice meant we let clients use some parts of the app without the server running, and updating the database with offline data when online again.

6.4 Dimensions and Areas

Figure 2 shows a simple building consisting of different built components. In this figure, the built components (coordinates are in two dimensions, and are not built components), are in one dimension. One dimension means we can have a length or a width or a height. We later describe how we can create an ontology from this in Figure 5. It shows that a Room individual can be left of, or right of another Room individual. Because of this, although there is no ordering of descriptive names of *one* dimension, it is easiest to understand it as length, which is also the convention to be used before width, and then height.

¹²<https://github.com/tiangolo/fastapi>

6.5 Privacy

Privacy is protected as much as possible in the implementation due to privacy concerns in tracking systems, especially in recent years. Data that can be used to identify an individual person is personal data [14, 27]. We implemented such that the personal data now is *pseudoanonymous* as well. This was done by assigning a new ID to the client every time the app was restarted as we don't need to permanently assign them. Only the ID and physical location are stored in Firebase, whilst the sensor data is stored for a maximum of thirty days in InfluxDB. More specifically, in Firebase endangered clients are stored by ID only, whilst IDs and the updated latitude and longitude coordinates are stored for the assigned clients. Note that there is no information on *when* the data was collected as this is handled by the server when writing to the real-time database.

Pseudoanonymous data still qualifies as personal data under GDPR (General Data Protection Registration) [46]. The data is not anonymous as the data is location dependent, and so movement pattern recognition can be used to correlate the data, and thus identify the individual data subjects (individuals).

The paragraph describing privacy in Section 2.6 states that no personal data should be collected without the permission of the user (data subject). Because of this, the implemented app requests the data subject for permission to access the physical location of the device running it. We realize that the specific purpose for data collection should be specified.

6.6 Discussion of Implementation

An alternative to the implemented digital twin (DT) is to make it wait a little while after it makes the informed decision and before it is sent to the physical counterpart, in which the smartphones are individually warned. One possible solution is to conclude that the user has not moved for a little while inside the critical area before the DT warns it. This is a limitation in this implementation as the user may have moved outside the critical area, meaning the informed decision could be misinforming the user. Another unexplored approach is to predict if the user will be inside a critical area.

The server reads from the knowledge graph and not the asset model. There is a limitation to how fast appropriate clients can be warned due to this, as described in Section 5.3 based on the descriptions in Section 2.4.1. We could warn the clients faster if the server automatically reads from the asset model at the expense of formalized knowledge of the domain. This could be implemented by adding the physical locations to the file from the SMOL program after smartphone objects are created, which the server would then access. But, this is also not currently possible in SMOL. The server implementation does not add the physical locations to the asset model because a time-series database should be used for this.

In addition to this, we tried to use the SMOL REPL as a Git submodule. The submodule includes the REPL-class definitions in which we can set settings automatically. We were able to use a dynamic asset model with it, and update

the twin, but not dump to the file¹³. A fully automatic system would benefit people in danger even more, and warn them faster, as discussed in Section 7.1. It took a long time to set it up due to using Maven, and with more time we would have fully automatic communication between the server and the twin as well.

In the server, there is no further reasoning done. As this could be considered a limitation, we should preferably also reason with the inbuilt tools from Apache Jena and OWL API. We did, however, use the reasoner tool HermiT in Protégé from the background information in Section 2.5 and in Figure 7, as well as automatically in SMOL, which showed consistency. And as the server queries what is already consistent, it was deemed unnecessary to reason further.

6.7 Discussion of Contributions

RQ1: *"How to create a dynamic asset model of a simple building that is also extensible?"*

We started with the background information on asset models and their benefits compared to other modeling tools. An ontology of a simple building was created and extended in the final implementation to support 2D areas. We also analyzed further what a dynamic asset model entails in Section 3.3. The result was an implementation supported by the research question. This was also evaluated to be the case in Section 5.2.2.

RQ2: *"How to enable bidirectional data flow between the PT and DT, so that the digital twin gets updated sensor data from mobile devices and sends informed decisions back?"*

From the background information on digital twins, as well as the realized architecture shown in Figure 12, we were able to implement seamless coupled communication between the PT and the DT. From the evaluation of the app, the appropriate clients are warned, even when they are many and heterogeneous. Value is derived from being warned as shown in Figure 18, where medical areas are marked in blue.

RQ3: *"How to separate dynamic data (smartphone's physical location) from static data (building infrastructure) in the asset model?"*

There was an in-depth analysis of this with illustrative examples in Chapter 3, and it was also discussed further in Section 6.1. The evaluation of the proposed implementation resulted in static data not being affected by dynamic data, as described in Section 5.2.1. The separation was also kept in mind when reloading the asset model with OWL API in the server implementation. Lastly, the static analysis of what a critical area entails is separated from the DT.

¹³<https://github.com/Owlar/Master-Thesis/tree/gradle-migration-try-fully-automatic>

7 Conclusion and Further Work

This chapter first discusses related work. Then contributions are presented in light of the research in this thesis. We conclude that we produced original work. Lastly, we suggest other directions to take in the future by using this thesis as a basis.

7.1 Related Work

There exist proposals relying on BIM (Building Information Modeling) for a digital twin (DT) in a real-world 2D environment (university building) [43]. But, BIM is expensive to model and maintain. It should be used to create a 3D replica, but to what extent when only 2D navigation is employed, and the set of movable assets change at any time, as described in Chapter 1, and evaluated in Section 5.2.1.

A goal must be to have interoperability between heterogeneous data sources in the domain. DTs should aim to offer even more context about a built environment by tracking assets from near-real time data [16]. We should aim for a web in which data is linked, as described in Section 2.2. We should encode human meaning in data, to ensure machines can read and understand what humans convey with the information. Chapter 5 demonstrated formalized knowledge, and interaction with a SPARQL endpoint.

One proposal links static data to dynamic data located elsewhere [62]. The separation can be done earlier in the asset model itself by examining how often data changes through time and preventing dynamic data from affecting static data when reloading it. Considering that various kinds of data are part of the domain, they should be separated early.

Last but not least, a proposal also uses a DT based on a model of a building (BIM in this case) in a crisis situation, only on one floor. More specifically, the scenario is of firefighters in a building that is on fire. Along with the model, the DT uses a shortest path algorithm to estimate oxygen required for a firefighter to safely exit the building [36]. In case of fire, there is a chance the provided exit routes are blocked, and thus a false positive is concluded. It is also stated that if a conclusion has been made by the DT, an alarm goes off. Assuming the alarm is heard by all firefighters, this approach yields a collaborative effort for decision making. But, it is not stated that the alarm stops ringing if the fire is put out, meaning it is a single bidirectional flow of data, whereas in this implementation there are endless loops. We let users navigate with Google maps, and medical areas are clearly shown in the app, as an alternative. In the context of a crisis situation we should not give the user too much information, but suggest safe approaches in an empathetic manner, as well as inform people on what to do *next*.

7.2 Contributions

From the research in this thesis, we developed a proof-of-concept implementation. It met all of the structural requirements presented in Table 2 besides one that was only met partly. We also conducted various real-life experiments in Chapter 5. To start off, we answer to the hypothesis (H) based on the discussion of the research questions (RQs) in Section 6.7.

H: *We can handle mobile assets in a semantic digital twin with the use of a dynamic asset model of a simple building, in which dynamic data (smartphone's physical location) is automatically updated by a server and separated from static data (existing building infrastructure), and informed decisions are sent back to the physical twin.*

Not only have all the RQs been successfully answered in Section 6.7, but the DT was evaluated to be a near-real time digital representation of the physical counterpart. 234 of 234 smartphones were concluded to be endangered as formalized knowledge. This real-life experiment at IFI shows that although the DT contains all mobile assets, it takes some time to finish writing approximately 14.000 lines to a file. Only one structural requirement was not met, as described in Section 4.3. The other requirements were met from discussion along the way in Chapter 4, and from evaluation in Chapter 5. Seamless bidirectional data flow was demonstrated in Section 5.2 based on the implementation in Section 4.2. Considering all of this and the descriptions in Section 7.1, this thesis produced original research. There were challenges along the way and things we could have done differently, as discussed in Section 6.6. One such thing is letting newly added smartphones be tracked as well, but all in all, we solved more than we initially thought was possible.

We present the following contributions:

- Formalized knowledge of very many mobile assets in a DT based on a dynamic asset model, in which static and dynamic data is separated early
- Endless loops of seamless bidirectional data flow between the PT and the DT, deriving value for people in danger
- An architecture for a digital twin model, in which the asset model is both inexpensive to maintain and extensible by manual operations

To conclude, a twinned system for tracking mobile assets based on a dynamic asset model, did not exist *until now*. The architecture of software components shown in Figure 12 was realized. The asset model is easily extensible by manual operators. It can inexpensively be maintained in an ontology editor, such as Protégé. In addition to this, the implementation was evaluated to separate static building data from dynamic data. By using semantic technologies we have semantically enriched the DT, and formalized the knowledge of the mobile assets inside it. This is ensured when the server automatically reads the knowledge graph. Value is derived from users being formally informed in the app, as shown

in Figure 18. Maintainers of the twin can easily use the SPARQL endpoint with informative messages, enabled by the SMOL REPL.

7.3 Future Work

There is other work that can be done by using the research in this thesis as a basis, and we hope it will be of value in the future. The points below are listed in an increasing level of difficulty and deviation from the work in this thesis.

- Based on the discussion in Section 6.1, future work could be done with the RDF graph produced from the asset model, or the knowledge graph for that matter. One could further evaluate how often data changes through time, and present it in a new graph using the gradient in Figure 37.
- Based on the discussion in Section 6.4, we only support two dimensions in this thesis. Future work could resolve around adding support for the third dimension height. An obvious limitation of this thesis is two areas (e.g. square rooms), one critical and one not, at different building levels with the same border corner points (latitude, longitude). However, adding support for height should not be very hard by using the work in this thesis. Nor would it be hard to check if smartphones are inside critical polygon areas by implementing the point-in-polygon problem in the twin. These areas could also be accurately shown in the app, letting users navigate out of them.
- We could implement proximity detection, to check how close smartphones are to other smartphones in the domain, or existing building infrastructure. It would be interesting to pinpoint the approximate location of the manually added smartphones based on the physical locations of the tracked ones. Getting informed decisions from other clients directly would make the the warning more efficient. It should not come at the expense of privacy, however, as discussed in Section 6.5.
- We could analyze data and implement anomaly detection with machine learning to check if the trends of physical locations abruptly and unrealistically change compared to previous movement patterns. Or use machine learning to predict if someone will be inside a critical area based on previous behavior, based on the discussion in Section 6.6.
- The work in this thesis can also be used by other disciplines, such as statistics, to study the data and draw conclusions. The conclusions can be studied in conjunction with the ones drawn by the DT from sufficient data. It would be interesting to look at the movement of movable entities through time whilst protecting privacy.

References

- [1] *5 Star Linked Data*. URL: https://www.w3.org/2011/gld/wiki/5_Star_Linked_Data (visited on 04/28/2023).
- [2] *A guide to GDPR data privacy requirements*. URL: <https://gdpr.eu/data-privacy/> (visited on 04/30/2023).
- [3] *A JSON-based Serialization for Linked Data*. URL: <https://www.w3.org/TR/json-ld11/> (visited on 04/27/2023).
- [4] Nusin Akram et al. “Design and Implementation of Asset Tracking System based on Internet of Things”. In: *2021 7th International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*. 2021, pp. 366–371. DOI: [10.1109/ICEEIE52663.2021.9616667](https://doi.org/10.1109/ICEEIE52663.2021.9616667).
- [5] *Apache Jena*. URL: <https://jena.apache.org> (visited on 04/30/2023).
- [6] Sean Bechhofer. “OWL: Web Ontology Language”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 2008–2009. ISBN: 978-0-387-39940-9. DOI: [10.1007/978-0-387-39940-9_1073](https://doi.org/10.1007/978-0-387-39940-9_1073). URL: https://doi.org/10.1007/978-0-387-39940-9_1073.
- [7] Thomas Bergs et al. “The Concept of Digital Twin and Digital Shadow in Manufacturing”. In: *Procedia CIRP* 101 (2021), pp. 81–84. ISSN: 2212-8271. DOI: <https://doi.org/10.1016/j.procir.2021.02.010>. URL: <https://www.sciencedirect.com/science/article/pii/S2212827121006612>.
- [8] Birgit Boss. *Digital Twins: the importance of semantic data structuring*. URL: <https://blog.bosch-digital.com/digital-twins-the-importance-of-semantic-data-structuring/> (visited on 04/22/2023).
- [9] *Digital Twins and Living Models at NASA*. URL: <https://ntrs.nasa.gov/citations/20210023699> (visited on 04/29/2023).
- [10] Fábio Matoseiro Dinis et al. “BIM and Semantic Enrichment Methods and Applications: A Review of Recent Developments”. In: *Archives of Computational Methods in Engineering* 29.2 (Mar. 2022), pp. 879–895. ISSN: 1886-1784. DOI: [10.1007/s11831-021-09595-6](https://doi.org/10.1007/s11831-021-09595-6). URL: <https://doi.org/10.1007/s11831-021-09595-6>.
- [11] *Firebase Realtime Database*. URL: <https://firebase.google.com/docs/database> (visited on 05/05/2023).
- [12] Aidan Fuller et al. “Digital Twin: Enabling Technologies, Challenges and Open Research”. In: *IEEE Access* 8 (2020), pp. 108952–108971. DOI: [10.1109/ACCESS.2020.2998358](https://doi.org/10.1109/ACCESS.2020.2998358).
- [13] Garcia-Molina et al. “Database Systems: The Complete Book”. In: Jan. 2002.

- [14] *GDPR personal data – what information does this cover?* URL: <https://www.gdpreu.org/the-regulation/key-concepts/personal-data/> (visited on 05/14/2023).
- [15] Birte Glimm et al. “HermiT: An OWL 2 Reasoner”. In: *Journal of Automated Reasoning* 53.3 (Oct. 2014), pp. 245–269. ISSN: 1573-0670. DOI: [10.1007/s10817-014-9305-1](https://doi.org/10.1007/s10817-014-9305-1). URL: <https://doi.org/10.1007/s10817-014-9305-1>.
- [16] Bjørn Godager, Erling Onstein, and Lizhen Huang. “The Concept of Enterprise BIM: Current Research Practice and Future Trends”. In: *IEEE Access* 9 (2021), pp. 42265–42290. DOI: [10.1109/ACCESS.2021.3065116](https://doi.org/10.1109/ACCESS.2021.3065116).
- [17] Michael Grieves and John Vickers. “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems”. In: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Ed. by Franz-Josef Kahlen, Shannon Flumerfelt, and Anabela Alves. Cham: Springer International Publishing, 2017, pp. 85–113. ISBN: 978-3-319-38756-7. DOI: [10.1007/978-3-319-38756-7_4](https://doi.org/10.1007/978-3-319-38756-7_4). URL: https://doi.org/10.1007/978-3-319-38756-7_4.
- [18] Grieves Michael. “Digital twin: manufacturing excellence through virtual factory replication”. In: *White Paper Series* 1.2014 (2014), p. 1. ISSN: 2194-1416.
- [19] Maren Parnas Gulnes. *Graph-based representation, integration, and analysis of neuroscience data*. en. 2020. (Visited on 11/04/2023).
- [20] Hesam Hamledari, Ehsan Azar, and Brenda McCabe. “IFC-Based Development of As-Built and As-Is BIMs Using Construction and Facility Inspection Data: Site-to-BIM Data Transfer Automation”. In: *Journal of Computing in Civil Engineering* 32 (Mar. 2018). DOI: [10.1061/\(ASCE\)CP.1943-5487.0000727](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000727).
- [21] Pascal Hitzler, Markus Krtzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. 1st. Chapman & Hall/CRC, 2009. ISBN: 1-4200-9050-X.
- [22] Aidan Hogan et al. “Introduction”. In: *Knowledge Graphs*. Cham: Springer International Publishing, 2022, pp. 1–4. ISBN: 978-3-031-01918-0. DOI: [10.1007/978-3-031-01918-0_1](https://doi.org/10.1007/978-3-031-01918-0_1). URL: https://doi.org/10.1007/978-3-031-01918-0_1.
- [23] Matthew Horridge. “A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3”. In: (Mar. 2011).
- [24] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. “Explaining Inconsistencies in OWL Ontologies”. In: *Scalable Uncertainty Management*. Ed. by Lluís Godo and Andrea Pugliese. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 124–137. ISBN: 978-3-642-04388-8.
- [25] *How Google uses location information*. URL: <https://policies.google.com/technologies/location-data?hl=en-US> (visited on 05/04/2023).

- [26] Zhisheng Huang et al. “Reasoning with Inconsistent Ontologies: Framework and Prototype”. In: (Jan. 2004).
- [27] *Hva er en personopplysning?* URL: <https://www.datatilsynet.no/rettigheter-og-plikter/personopplysninger/> (visited on 05/14/2023).
- [28] *InfluxDB*. URL: <https://www.influxdata.com/> (visited on 05/05/2023).
- [29] *Introduction — The SMOL Language documentation*. URL: <https://smolang.org/introduction.html#introduction> (visited on 01/05/2023).
- [30] *Ionic*. URL: <https://ionicframework.com> (visited on 08/05/2023).
- [31] Eduard Kamburjan et al. “Digital Twin Reconfiguration Using Asset Models”. In: *Proc. 11th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2022)*. Vol. 13704. Lecture Notes in Computer Science. Springer, 2022, pp. 71–88.
- [32] Eduard Kamburjan et al. “Programming and Debugging with Semantically Lifted States”. In: *Proc. 18th International Conference on the Semantic Web (ESWC 2021)*. Ed. by Ruben Verborgh et al. Vol. 12731. Lecture Notes in Computer Science. Springer, 2021, pp. 126–142.
- [33] Eduard Kamburjan et al. “Twinning-by-Construction: Ensuring Correctness for Self-Adaptive Digital Twins”. In: *Proc. 11th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2022)*. Vol. 13701. Lecture Notes in Computer Science. Springer, 2022, pp. 188–204.
- [34] Suneuy Kim et al. “GeoYCSB: A Benchmark Framework for the Performance and Scalability Evaluation of Geospatial NoSQL Databases”. In: *Big Data Research* 31 (2023), p. 100368. ISSN: 2214-5796. DOI: <https://doi.org/10.1016/j.bdr.2023.100368>. URL: <https://www.sciencedirect.com/science/article/pii/S2214579623000011>.
- [35] Werner Kritzinger et al. “Digital Twin in manufacturing: A categorical literature review and classification”. In: *IFAC-PapersOnLine* 51.11 (2018), pp. 1016–1022. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2018.08.474>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896318316021>.
- [36] Leucker, Martin, Sachenbacher, Martin, and Vosteen, Lars Bernd. “Digital Twin for Rescue Missions - a Case Study”. In: pp. 28–38. URL: <https://www.duo.uio.no/handle/10852/101662> (visited on 05/23/2023).
- [37] Luning Li et al. “Digital Twin in Aerospace Industry: A Gentle Introduction”. In: *IEEE Access* 10 (2022), pp. 9543–9562. DOI: [10.1109/ACCESS.2021.3136458](https://doi.org/10.1109/ACCESS.2021.3136458).
- [38] Marco Macchi et al. “Exploring the role of Digital Twin for Asset Lifecycle Management”. In: *IFAC-PapersOnLine* 51.11 (2018), pp. 790–795. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2018.08.415>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896318315416>.

- [39] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. “Leveraging Digital Twin Technology in Model-Based Systems Engineering”. In: *Systems* 7.1 (2019). ISSN: 2079-8954. DOI: [10.3390/systems7010007](https://doi.org/10.3390/systems7010007). URL: <https://www.mdpi.com/2079-8954/7/1/7>.
- [40] *Maps SDK*. URL: <https://developers.google.com/maps/documentation/android-sdk/overview> (visited on 05/04/2023).
- [41] Noorlela Marcheta et al. “Development of Mobile Asset Management Applications with Geolocation and Haversine”. In: *2022 5th International Conference of Computer and Informatics Engineering (IC2IE)*. 2022, pp. 1–5. DOI: [10.1109/IC2IE56416.2022.9970187](https://doi.org/10.1109/IC2IE56416.2022.9970187).
- [42] Mohamed Mohamed, Obay Altrafi, and Mohammed Ismail. “Relational Vs. NoSQL databases: A survey”. In: *International Journal of Computer and Information Technology (IJCIT)* 03 (May 2014), p. 598.
- [43] Pieter Pauwels et al. “Live semantic data from building digital twins for robot navigation: Overview of data transfer methods”. In: *Advanced Engineering Informatics* 56 (2023), p. 101959. ISSN: 1474-0346. DOI: <https://doi.org/10.1016/j.aei.2023.101959>. URL: <https://www.sciencedirect.com/science/article/pii/S1474034623000873>.
- [44] Petroc Taylor. *Number of smartphone mobile network subscriptions worldwide from 2016 to 2022, with forecasts from 2023 to 2028*. 2023. URL: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/> (visited on 04/14/2023).
- [45] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell*. In a Nutshell (o’Reilly) Series. O’Reilly Media, 2005. ISBN: 978-0-596-00795-9. URL: <https://books.google.no/books?id=Xg8wLPmt5CMC>.
- [46] *Pseudonymous data: processing personal data while mitigating risks*. URL: https://edps.europa.eu/press-publications/press-news/blog/pseudonymous-data-processing-personal-data-while-mitigating_en (visited on 05/14/2023).
- [47] *RDF and Change over Time*. URL: <https://www.w3.org/TR/rdf11-concepts/#dfn-rdf-vocabulary> (visited on 02/05/2023).
- [48] *Resource Description Framework (RDF)*. URL: <https://www.w3.org/RDF/> (visited on 04/19/2023).
- [49] Vetle Ryen, Ahmet Soylu, and Dumitru Roman. “Building Semantic Knowledge Graphs from (Semi-)Structured Data: A Review”. In: *Future Internet* 14.5 (2022). ISSN: 1999-5903. DOI: [10.3390/fi14050129](https://doi.org/10.3390/fi14050129). URL: <https://www.mdpi.com/1999-5903/14/5/129>.
- [50] Gustaph Sanga and Xingxing Hao. “A Spectral Clustering-Based Community Detection Algorithm for RDF Graphs”. In: *2020 2nd World Symposium on Artificial Intelligence (WSAI)*. 2020, pp. 18–23. DOI: [10.1109/WSAI49636.2020.9143285](https://doi.org/10.1109/WSAI49636.2020.9143285).

- [51] *Semantic Access*. URL: <https://smolang.org/language/semantic-access.html> (visited on 01/05/2023).
- [52] *Semantic Web Standards*. URL: https://www.w3.org/2001/sw/wiki/Main_Page (visited on 04/24/2023).
- [53] *Shapes Constraint Language (SHACL)*. URL: <https://www.w3.org/TR/shacl/> (visited on 04/26/2023).
- [54] *SPARQL By Example: The Cheat Sheet*. URL: https://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf (visited on 08/08/2023).
- [55] Peter Szolovits, Lowell B. Hawkinson, and William A. Martin. “An Overview of OWL, a language for knowledge representation”. In: 1977.
- [56] *Terse RDF Triple Language*. URL: <https://www.w3.org/TR/turtle/> (visited on 02/05/2023).
- [57] *The OWL API*. URL: <https://owlapi.sourceforge.net> (visited on 04/30/2023).
- [58] *The SMOL Language*. URL: <https://smolang.org/> (visited on 04/21/2023).
- [59] Tim Berners-Lee. *Linked Data*. URL: <https://www.w3.org/DesignIssues/LinkedData.html> (visited on 04/24/2023).
- [60] *Time Series Access*. URL: <https://smolang.org/language/semantic-access.html#time-series-access> (visited on 05/05/2023).
- [61] *Triple Store*. URL: <https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html> (visited on 02/05/2023).
- [62] Maryna Waszak et al. “Let the Asset Decide: Digital Twins with Knowledge Graphs”. In: *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. 2022, pp. 35–39. DOI: [10.1109/ICSA-C54293.2022.00014](https://doi.org/10.1109/ICSA-C54293.2022.00014).
- [63] *What is time series data?* URL: <https://www.influxdata.com/what-is-time-series-data/> (visited on 05/05/2023).