Vrije Universiteit Brussel

Faculty of Sciences and Bioengineering Sciences
Department of Computer Science

Interoperable and Discoverable Indoor Positioning Systems

Author:
Maxim Van de Wynckel

Dissertation presented in fulfilment of the
requirements for the degree of Doctor of Sciences

Members of the Jury

Prof. em. Dr. Olga De Troyer	*Vrije Universiteit Brussel (Chair)*
Prof. Dr. Elisa Gonzalez Boix	*Vrije Universiteit Brussel (Secretary)*
Prof. Dr. Beat Signer	*Vrije Universiteit Brussel (Promoter)*
Prof. Dr. Bart Jansen	*Vrije Universiteit Brussel*
Prof. Dr. Pieter Colpaert	*Ghent University*
Prof. Dr. Kris Luyten	*Hasselt University*

VUB VRIJE UNIVERSITEIT BRUSSEL

Interoperable and Discoverable Indoor Positioning Systems

Publisher: Vrije Universiteit Brussel – WISE Lab

ISBN: 978-90-835689-0-4
NUR-code: 980

Abstract

In the early days of navigation, humans relied on visual landmarks such as mountains, coastlines, and structures to determine their location and direction. With the arrival of tools like compasses and sextants, navigation became more precise, enhancing navigation and positioning. In the modern era, outdoor navigation relies heavily on global satellite-based positioning systems such as GPS, allowing individuals to navigate around the globe, locate points of interest, and assist robots in positioning themselves in both known and unknown environments.

While indoor positioning systems have been in development for decades, their adoption has accelerated in recent years in complex environments like airports, supermarkets, and warehouses. These systems are employed for tasks such as resource tracking, automated order picking, and guiding users through complex building layouts. Unlike traditional positioning systems, which rely on a fixed set of techniques, indoor positioning or navigation systems often use diverse technologies and hardware, usually requiring proprietary software or intricate calibration to function. This reliance on proprietary solutions creates several challenges. Users are often required to install software for each specific location or building, fragmenting the user experience and raising concerns about privacy and data ownership. For building owners, the high costs and complexity of developing and deploying these systems limit broader adoption. Moreover, closed systems hinder interoperability, making it difficult to integrate or share data across different platforms.

These challenges extend beyond positioning systems and reflect general issues with user data, where closed applications restrict seamless data exchange from one platform to another. One promising solution involves creating open frameworks that enable seamless data interoperability, giving users greater control over their information while reducing dependence on systems owned and managed by a single organisation. Although regulatory initiatives within the European Union are gradually enabling data interoperability, significant obstacles remain, particularly in domains where data does not rely on existing standardisations, such as is the case with indoor positioning systems.

This research focuses on enabling seamless data and knowledge exchange between positioning systems, both indoor and outdoor. It addresses the limitations of current approaches by proposing a framework that facilitates the integration, discovery, and management of location data in a way that prioritises user control and interoperability. Central to this research is the development of improved vocabularies that enhance machine understanding of location data, providing a foundation for seamless communication between systems. Additionally, it explores techniques for enabling positioning systems to discover one another without prior coordination, thereby reducing the dependency on pre-established connections and broadening their use and reuse in other use cases. The research also introduces strategies for managing user data effectively, ensuring privacy and flexibility while supporting seamless exchange across systems. These contributions collectively advance the development of an open, interoperable ecosystem for both indoor and outdoor positioning systems, addressing key limitations of current approaches and paving the way for a future where location data can be stored, shared, and accessed effortlessly across a wide range of use cases and technologies.

Acknowledgements

When I first started my PhD, I already had many ideas I wanted to realise. In my mind, I would have plenty of time to realise those dreams. It did not take long to find out that with each completed idea, several new ones pop up. First of all, I want to express my sincere gratitude to my **supervisor** Prof. Dr. Beat Signer. I first met Beat a week before submitting my Master's thesis. I left that first meeting with a hundred new ideas that I still wanted to implement for my thesis. At that point, I realised that the *wise* decision would be to start my new journey as a PhD student at the Web & Information Systems Engineering (WISE) lab to continue exploring those ideas. He helped to steer those ideas in the right direction and helped to manage my ideas in a pragmatic approach that I will continue to use throughout the rest of my career.

I would also like to thank the **members of the jury**: Prof. Dr. Kris Luyten, Prof. Dr. Pieter Colpaert, Prof. Dr. Bart Jansen, Prof. Dr. Elisa Gonzalez Boix and Prof. em. Dr. Olga De Troyer for their time, efforts and input. Their valuable feedback and discussions helped improve the final version of this dissertation.

Next, the **colleagues** that I have met during this PhD — both within the lab, around the VUB and at conferences, really helped to encourage me to continue with my journey. I have met great people from various communities who helped me realise which directions still required more research.

I was lucky that the VUB has its own diving club, which I immediately joined in my first week. I want to thank my friends at the **V.U.B. Diving Center** for helping me take my mind off things. Each training and each dive were a welcome break from my research. Following, I also want to express my gratitude to my **family** for their support throughout this journey. Their encouragement and understanding made it easier for me to focus.

Finally, last but definitely not least, I would like to thank **my wife**, Emmelien, for her support both at the start of my PhD and during. It was a long journey filled with ups and downs, but she was always there to support me and keep me going.

Contents

Acronyms

NGSI-LD	Next Generation Service Interfaces – Linked Data
A-GPS	Assisted GPS
ACL	Access-Control List
AoA	Angle of Arrival
AoD	Angle of Departure
AP	Access Point
API	Application Programming Interface
AR	Augmented Reality
BLE	Bluetooth Low Energy
CAGR	Compound Annual Growth Rate
CDAE	Collaborative Deep Denoising Autoencoder
CI	Cell Identification
CNN	Convolutional Neural Network
CORS	Cross-Origin Resource Sharing
CRF	Coordinate Reference Frame
CRS	Coordinate Reference System
DGA	Data Governance Act
DHT	Distributed Hash Table
DMA	Digital Markets Act
DNS	Domain Name System
DTR	Digital Twin Registry
ECEF	Earth-centered Earth-fixed
EDPS	European Data Protection Supervisor
ESWC	Extended Semantic Web Conference
EU	European Union
EWKT	Extended Well-known Text Representation

FAIR	Findable, Accessible, Interoperable and Reusable
FOSDEM	Free and Open source Software Developers' European Meeting
FPS	Frames per Second
GCS	Geodetic Coordinate System
GDPR	General Data Protection Regulation
GLONASS	Globalnaya Navigatsionnaya Sputnikovaya Sistema
GML	Geography Markup Language
GMLC	Gateway Mobile Location Centre
GMS	GeoMobility Server
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ILBS	Infrastructure-providing LBS
IMDF	Indoor Mapping Data Format
IMU	Inertial Measurement Unit
IoT	Internet of Things
IPIN	Indoor Positioning and Indoor Navigation
IPS	Indoor Positioning System
ISWC	International Semantic Web Conference
J2ME	Java 2 Micro Edition
JS	JavaScript
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
LAN	Local Area Network
LBS	Location-based Service
LDAP	Lightweight Directory Access Protocol
LDES	Linked Data Event Stream
LDF	Linked Data Fragment
LDHT	Linked Data Hash Table
LDN	Linked Data Notifications
LDP	Linked Data Platform
LDR	Linked Data Resource

LiDAR	Light Detection and Ranging
LIF	Location Interoperability Forum
LOT	Linked Open Terms
LPPM	Location Privacy Protection Mechanism
MLP	Mobile Location Protocol
MPC	Mobile Positioning Center
MQTT	Message Queuing Telemetry Transport
MTMCT	Multi-Target Multi-Camera Tracking
NAD	North American Datum
ODBC	Open Database Connectivity
ODM	Object-Document Mapping
OGC	Open Geospatial Consortium
OSM	OpenStreetMap
OUPnP	Outdoor Universal Plug and Play
P2P	Peer-to-Peer
PDR	Pedestrian Dead Reckoning
PDV	Personal Data Vault
PII	Personally Identifiable Information
POSO	Positioning System Ontology
QR	Quick Response
QUDT	Quantity, Unit, Dimension and Type
RDF	Resource Description Framework
REGEX	Regular Expression
RF	Radio Frequency
RFID	Radio Frequency Identification
ROS	Robot Operating System
RPC	Remote Procedure Call
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
S3DB	Simple 3D Buildings
SDK	Software Development Kit
SHACL	Shapes Constraint Language
ShEx	Shape Expressions
SIT	Simple Indoor Tagging

SLAM	Simultaneous Localisation and Mapping
SLP	Service Location Protocol
SMA	Simple Moving Average
Solid	Social Linked Data
SOSA	Sensor, Observation, Sample and Actuator
SOTA	State of the Art
SPARQL	SPARQL Protocol and Query Language
SRSI	Spatial Reference System Identifier
SSID	Service Set Identifier
SSN	Semantic Sensor Network
TDoA	Time Difference of Arrival
ToA	Time of Arrival
TOGAF	The Open Group Architecture Framework
UAC	User Access Control
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
UWB	Ultra-wideband
UX	User Experience
VR	Virtual Reality
VSDS	Vlaamse Smart Data Space
W3C	World Wide Web Consortium
WAC	Web Access Control
WebID	Web Identifier
WGS	World Geodetic System
WKT	Well-known Text Representation
WLAN	Wireless LAN
WoT	Web of Things
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

> *The problem with apps, and by this I mean native apps that must be downloaded to your phone, is that they are just becoming too much trouble to organize and maintain. It's just not realistic to have an app for every store you go to, every product you own and every website you visit. This creates an ever increasing set that must be curated, organized and culled. (...) If you were to walk into a store and they proudly proclaimed on the door that they had an app, would you immediately install it? What value/pain calculation would you perform in your head before going through the trouble? (...) My point is that if somehow the app magically appeared on your phone as you walked in the door would you be more likely to try it?*
>
> – Scott Jenson (2011) [1]

I have probably entered thousands of buildings in my life. From what I assume, most of those buildings did not have a system in place that could track me or that allowed me to navigate inside — but I cannot be certain about this. Even if complex buildings such as a hospital, airport, conference hall, supermarket or university have a mobile application to track my location, I am not confident that I would have the incentive to search for it and install it.

The quote by Scott Jenson, a UX designer who worked at Google and Apple, relates to the issue of indoor positioning and navigation systems. If you were to walk into a building and they advertise on the door that they have an application to help you find your way, would you install it? More importantly, would you still remember that there is an application available if you get lost within the building? On the flip side, what motivates building owners to develop their own navigation applications for visitors?

When navigating to a specific destination or tracking the location of our food being delivered, we often use services that can track the position of the *thing* we are tracking. Outdoors, these services primarily rely on the Global Positioning System (GPS), which is a collection of satellites that help us to determine an object's position on the surface of the Earth. While satellite-based positioning systems such as GPS are common, they are not the only technology available to determine an object's position. Inside a building, underground, or in a forest, we have to rely on other technologies due to the inaccuracy of GPS signals when there is no direct line of sight to satellites. Even outdoors, we often resort to other techniques in use cases where a more accurate and reliable position is needed.

If we focus on use cases for tracking a position inside a building, we can list examples such as indoor navigation, asset tracking, and proximity marketing. To compute this type of position, various technologies such as Bluetooth beacons, Wi-Fi, Ultra-wideband (UWB), and Radio Frequency Identification (RFID) have been utilised to replace satellite-based positioning systems. These technologies have different advantages and limitations that must be considered when choosing the most suitable solution for a particular use case. The systems which are deployed in an indoor environment to track persons or objects are often called indoor positioning systems (Definition 1). Research in the field of indoor positioning systems is still ongoing, with a focus on improving the accuracy, scalability, and robustness of these systems for widespread adoption in various industries. With indoor environments varying from building to building, different technologies might perform better or worse in specific environments, preventing the use of a single technology that applies to all indoor environments. Furthermore, while satellite-based positioning systems only require a fixed set of satellites to cover a large area, the same cannot be said for indoor environments, which often require different hardware in each building, making their implementation slow and expensive.

Definition 1: Indoor Positioning System

A positioning system that is used indoors to provide use cases such as indoor position tracking or indoor navigation is defined as an indoor positioning system (IPS) [2].

This lack of standardisation causes a spread of different technologies across buildings and positioning systems, causing each Indoor Positioning System (IPS) to be an individual implementation or at least an implementation based on a framework that defines the technologies and algorithms. In turn, this causes the spread of data and knowledge over proprietary systems and applications. To keep this valuable data relevant to the user or object that is being tracked, we need a solution that can bridge the gap between individual positioning systems and between the user and the positioning system.

Moreover, users also own more devices that are capable of tracking them. Smartwatches, augmented reality (AR) headsets and even self-driving cars are types of devices that obtain a position. In turn, these devices collect tracking data of a single user, yet the data is often fragmented across different services for those specific devices. In addition to bridging the gap between positioning systems, we also need to find a user-centric solution to manage the sources which produce this data.

Fragmentation can be addressed by standardisation [3]. However, due to the lack of standardisation and lack of technologies that could potentially serve as a single standard, this fragmentation must be solved by enabling the seamless exchange of data. By doing so, building owners would no longer need to focus on the design of mobile applications or tracking software, and could instead rely on existing solutions that can be integrated with their setup. This would not only significantly reduce costs but also enable users to decide which solutions they use for tracking, allowing smaller businesses to reach a broader audience.

1.1 Privacy and Transparency Problems

One major concern with the tracking of users relates to the privacy implications of this data. In the first month of 2025 alone, multiple cases of leaked location data were reported, ranging from mobile applications leaking accurate GPS data[1] to the location of parked electric cars[2].

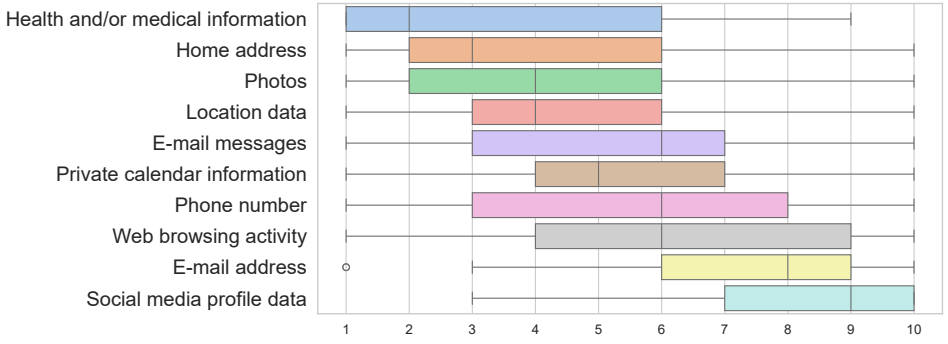


Figure 1.1: Ranking of personal data according to our survey

In a research study that we conducted in 2025 (see Appendix E and [4]), we have let users rank different types of personal data in order of importance. Figure 1.1 illustrates these results, showing users consider location data the fourth most important type of data for users. A home address is considered the second most important

[1] https://www.theverge.com/2025/1/13/24342694/gravy-analytics-location-data-broker-breach-hack-disclosed

[2] https://www.theverge.com/2024/12/30/24332181/volkswagen-data-leak-exposed-location-evs

type of personal data for users, which is inferable from a user's (or parked car's) location. These results align with earlier work where users indicated that location data is a valuable type of data [5, 6].

On the other side of the valuation of location data, companies also value this data to obtain knowledge about the behaviour or expected behaviour of customers [7, 8]. Having the location data spread over multiple closed databases and companies not only prevents users from managing and accessing their own location data but also requires the creation of proprietary applications to access these databases.

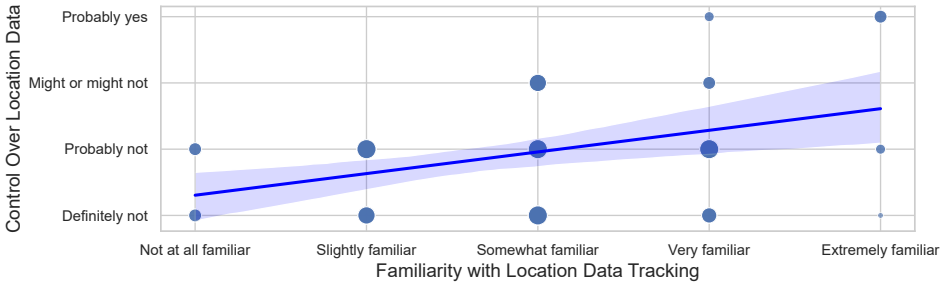


Figure 1.2: Correlation between the familiarity with location data tracking and user-perceived control of location data

In our study, most participants indicated they feel they do not have enough control over their location data. In Figure 1.2, we see a clear correlation between the expertise of the participant and their perception of how much control they have. Participants who indicated that they do not have enough control, indicated that they feel violated when services can obtain this personal information. They do not want to share data without explicit consent, and generally are unaware of how this data is used and how it will prevent services from operating when disabling location sharing. Several participants who were familiar with location tracking concepts indicated that they take measures to prevent tracking, often resorting to different privacy-aware operating systems or applications.

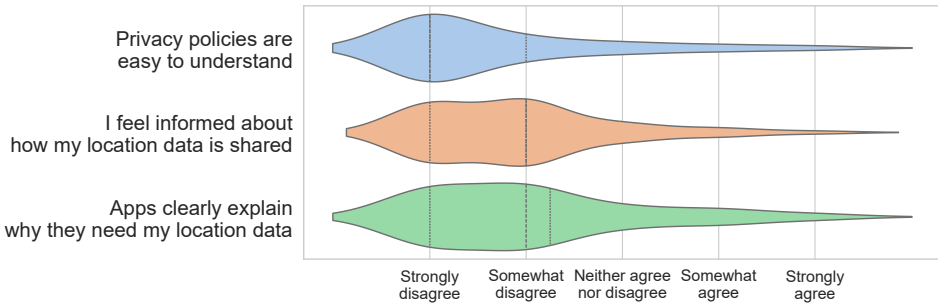


Figure 1.3: Transparency of applications and systems about location data usage

Legislations in Europe, such as the General Data Protection Regulation (GDPR) [9] or the Data Governance Act (DGA) [10] aim to protect personal data and ensure that individuals are informed and have control over how their information is used and shared. Privacy policies aim to provide insight into what data is processed and how it is used. Still, as illustrated in Figure 1.3, the majority of users find these privacy policies difficult to understand and do not feel informed about how their location data is used. What is more, only 50.6% of our participants reported that they read a privacy policy when installing an application. In total, we had 58 participants, 89.7% of these participants were from the European Union.

1.2 Towards FAIR Positioning Systems

Market studies [11, 12, 13] estimate the current indoor positioning and indoor navigation market around 20 billion USD with a compound annual growth rate (CAGR) growth between 21.4% and 37.6% in the next five years. More buildings will deploy indoor positioning or navigation applications to some extent, either to provide navigation assistance to visitors, positioning autonomous robots, AR applications, or to collect data on foot traffic patterns for business optimisation.

In 2016, the FAIR guiding principles were published [14] to promote the findability, accessibility, interoperability, and reusability of research data. Ever since, these principles have found their way to other domains, including Global Navigation Satellite Systems (GNSS) [15]. For indoor positioning systems, the data and systems themselves remain *unFAIR*. With data sovereignty such as the Digital Markets Act (DMA) [16] in mind, the field of indoor positioning systems should be prepared to offer *interoperable and discoverable indoor positioning systems* to adhere to these regulations.

Definition 2: Interoperability

Interoperability is the ability of computer systems or programs to seamlessly exchange information in a way that can be accessed, read and understood by other systems [17, 18].

Interoperability, which forms part of the FAIR principles, is the ability of a computer system or program to seamlessly exchange information in a way that can be accessed, read and interpreted by other systems [17, 18]. In our context of interoperable positioning systems, it entails the ability for different systems, services and applications to work together seamlessly without requiring individual positioning systems to interface with one another. In a perfect scenario, if each positioning system handled data in a way that enables the seamless exchange of this data, it would allow users to use a single navigation application for outdoors, indoors or other

environments that track a location. Furthermore, users would have more control over how data produced by one system is reused in another system or application.

Definition 3: Discoverability

Discoverability or findability of a service is the ability for users to find and retrieve information on how to interface with a service.

At the beginning of this introduction, we presented a quote from Scott Jenson. It was written at the time when he pitched the concept of the Physical Web. The idea of the Physical Web was that a single application, in this case, a web browser, would be able to discover and interact with businesses, devices and people within proximity. One of the challenges we wish to solve is how users can discover or find positioning systems within an environment. This discoverability plays an important role when working with interoperable systems. One can have as many systems and applications that work together with the same data, but if users cannot discover the availability of this data or services, they ultimately become useless.

1.3 Problem Statement

Unlike outdoor positioning systems, Indoor Positioning Systems (IPSs) do not rely on a single standardisation of the technologies used to implement such a system. ISO standards such as ISO 17438:2024 [19, 20, 21] and ISO 18305:2016 [22] define the basic data models and technologies of current IPSs. However, with rapid advances in the domain, these individual standardisations are also rapidly outdated to serve as a single standard for each positioning system.

Surveys and other related work also acknowledge this issue of non-standardised IPSs. In [23], the authors indicate a *"Lack of Standard and Interoperability"* as one of the main challenges in the development of indoor positioning systems. They emphasise the problem of the lack of standards in the used technologies, since there are currently no technologies that would serve as a general standard for different indoor environments and use cases. Deng et al. mention *"In different industrial applications, most companies will adopt the specialised standards common in that industry, resulting in a waste of resources that impedes sharing and interoperability"* [24]. In their work, they indicate that some standardisations exist for indoor positioning services, but they are often focused on particular goals and require complementary unstandardised work that impedes overall interoperability. Similar to Deng et al., Furfari et al. [25, 26] also indicate that the most important issue of indoor positioning systems is the lack of methods for integrating and enabling cooperation between such systems. They highlight the need for standardisation in both the discovery and communication of these systems, but agree that in the current industry, one single all-encompassing standardisation does not exist.

> *Every actor adopts its own solution in terms of positioning technologies and data processing and, more importantly, such systems are not designed to cooperate. As a result, we expect that the market for indoor systems will become quickly **fragmented** and characterized by **incompatible** solutions.*
>
> – Furfari et al. (2021) [26]

Lack of standardisation and interoperability results in a fragmentation of user data across services and proprietary applications. This is not only true for indoor systems, but any non-interoperable or non-standardised system [27]. Due to the vast amount of building-specific implementations, users are forced to find and use specific services and applications for navigating in a particular building, but how will they find these services?

During our research, while addressing the lack of interoperability, we noticed that the design of an interoperable system does not immediately solve market fragmentation. Similar to related issues of data and service fragmentation [28], the discovery of interoperable systems is a needed step towards solving this fragmentation. Ultimately, these problems stem from the *unFAIR* IPSs currently in use, where data is not accessible, applications are not reusable and the systems themselves are not findable.

Based on the list of problems from the aforementioned related work, our own observations in the domain and similar general issues of data fragmentation in different domains, we derived the following three main problems to be addressed in our research:

Problem 1 **Lack of standardisation and interoperability between indoor positioning systems**
In line with related work [24, 25, 23], we agree that the lack of technology standardisation is one of the main reasons why indoor positioning systems are often designed as proprietary solutions managed by a single organisation. This results in a lack of interoperability between such systems, as their underlying technologies and implementations vary significantly.

Problem 2 **Lack of access to the data produced and consumed by indoor positioning systems and applications**
Continuing from Problem 1, the lack of data interoperability results in the fragmentation of user data across different services. Services that want to (re)use data currently have no access to this data unless specific interfaces are developed.

Problem 3 **Fragmented and non-reusable indoor positioning systems and applications**
Due to the lack of interoperability (Problem 1) and the lack of access to data (Problem 2), indoor positioning systems are fragmented across multiple services and applications. As a user, it is challenging to navigate different buildings with different indoor positioning systems, as each system may require a separate app or device for navigation. This lack of cross-building indoor positioning systems hinders the collaboration between IPSs and the reusability of indoor positioning technologies.

1.4 Research Questions

Whenever we combine multiple technologies for positioning, we define the positioning system as an integrated or hybrid positioning system as described in Definition 4. Such a system takes input data from one or more positioning techniques and outputs certain data that can be used for further processing. Indoor positioning systems are often hybrid systems that are tailored to an indoor environment. In our research, we consider hybrid positioning systems as the general type of positioning system we want to support, since they are broad enough to be used in different use cases.

Definition 4: Hybrid Positioning System

A hybrid or integrated positioning system is a positioning system that integrates multiple positioning technologies and techniques to overcome the limitations of individual systems and enhance overall positioning performance [29].

One of the first challenges in designing interoperable systems is to ensure the semantic interoperability of the data used by such systems. In the case of a hybrid positioning system, this includes the data that the system requires from various sources, as well as the data it produces. With hybrid systems being capable of combining various techniques, we need to generalise the data of a hybrid positioning system so that we can generalise indoor positioning systems. Based on the previously identified problems, we formulated the following research questions to guide our research:

RQ1 **How can we represent hybrid positioning systems?**
We want to achieve interoperable indoor positioning systems that are future-proof in the landscape of unstandardised technologies. In order to achieve this, we first need to represent the concept of positioning systems, and more specifically, hybrid positioning systems that combine multiple technologies.

Representing a hybrid positioning system involves a generic data and processing representation of how such a system works and is constructed. At the same time, we also need to generalise and define the data that these positioning systems require and produce. By representing a hybrid positioning system rather than an indoor system, we ensure that our question is broad enough to cover a wide range of use cases, technologies and implementations.

The representation refers to the way data is stored and processed within a single system, while interoperability (as detailed later in RQ2) focuses on how different systems can seamlessly exchange information. The data representation we aim to obtain by answering this research question can be tailored to a single system, but it must be capable of representing different systems, algorithms, technologies and use cases. Next, we can enable the data exchange by utilising this representation.

RQ1.1 How can we generalise the input of a positioning system?
For a positioning system to be able to compute data, the input data needs to be in a known format and structure. Various existing technologies have their own data format. We need to generalise the input of the data in a scalable structure that can be applied to a wide range of technologies.

RQ1.2 How can we generalise the output of a positioning system?
After processing the input data, a positioning system provides some output data. While we can assume that this most likely includes a position, it can also include other data, such as orientation, velocity or even a map of the environment. The output needs to be generalised in a way that makes it clear what the output can contain after being processed by a wide range of different techniques or algorithms. We want to generalise the structure, granularity and type of this data so, together with RQ1.1, we can represent a complete positioning system. In RQ2, we will make use of this data representation to enable the exchange between systems.

RQ2 How can we create interoperable positioning systems?
Interoperability was described as the ability to read, understand and access data across different systems (Definition 2). With RQ1 and its sub-questions, we investigate a data representation to be used within a single system. When creating interoperable positioning systems, we aim to develop systems that allow data to be exchanged between systems without losing any context. If we look a bit further at *processing interoperability* [30], we also want to be able to understand how this data is processed. In RQ1, we generalise indoor positioning systems by representing hybrid systems, but do not focus on the seamless data exchange between systems.

RQ2.1 How can we enable access to the data of a positioning system?
Enabling access to the data of a positioning system is not as easy as making the information publicly available. Positioning systems work with sensitive user data, which should only be accessible by applications the user trusts. In this research question, we will investigate the possibilities for users to enable access to their data, allowing them to decide which positioning systems or applications can access and update their location data.

RQ2.2 How do we avoid ambiguity and enable semantic reasoning on the data of a positioning system?
To satisfy the ability to understand the data, semantic reasoning should be possible on the positioning data. Semantic reasoning is the ability of a system to infer information that is not explicitly defined within the data [31]. Such inference is necessary when working with data of unknown origin. In this research question, we will investigate how we can provide processing interoperability so that other systems are aware of how a positioning system processes information. This, in turn, can provide additional context on the quality of the data produced by such a system.

As an interoperable system, one can have access to read data and the semantic knowledge to understand the data. However, understanding goes beyond being able to interpret information. In an interoperable system, the interpretation of the data needs to be unambiguous for two systems to interpret the data in the same way.

RQ3 How can we discover positioning systems?
Creating interoperable positioning systems that can be accessed is important. However, when users and services are unaware of the existence of these systems, they become useless. Discovery is a broad term that encompasses the process of finding and identifying positioning systems that are available for use. To answer this research question, we will explore methods and technologies that enable the discovery of positioning systems to ensure that users and services are aware of their existence and capabilities.

RQ3.1 How can we ensure interoperability in the discovery of a positioning system?
Discovering a system or service often requires intricate and tailored procedures. We want to ensure the accessibility and readability of a system during this discovery. However, we aim to prevent the use of discovery protocols in interoperable systems that require additional hardware beyond existing infrastructure.

RQ3.2 How do we discover a positioning system in a physical environment?
The discovery can happen using a centralised service or an interconnected decentralised network. However, when we want to discover a service bound to a geographical space, we also want to discover this service within the physical world. Taking research question RQ3.1 into account, in this research question, we focus on the physical discovery of online services.

1.5 Objectives

Based on the research questions outlined in the previous section, we defined the following five objectives to answer these research questions. These objectives guided our research and enabled us to come up with a solution for the three main problems with indoor positioning systems mentioned earlier.

Objective 1 Investigate positioning systems and generalise the data and processing flow of such systems to align with a wide range of use cases
Our first objective is to address RQ1 by investigating different positioning systems, frameworks and technologies. Based on this, we aim to create a data and processing representation of hybrid positioning systems that aligns with various use cases and technologies.

Objective 2 Design a solution to enable the semantic description of the generalised parts of a positioning system and its data
In the previous objective, we design a generalised data and processing flow representation for positioning systems. In Definition 2, we defined interoperability as the ability for computer systems to exchange information in a way that can be read and understood. To enable interoperability, we need to semantically describe the positioning system, its processing flow and the data it produces to provide other systems with enough context to reason about the data (as defined with RQ2.2).

Objective 3 Investigate and design a solution for discovering and enabling access to the data produced by a positioning system
With the previous objective, we aim to provide a solution to create semantically described positioning systems and their data. The discovery of the data produced by such a system, as well as the access to this data, must still be addressed. With this objective, we plan to answer RQ2.1.

Objective 4 **Investigate how to enable users to discover the existence of indoor positioning systems and how to interface with them**
In the current landscape of fragmented indoor positioning applications, implementing various use cases or developing implementations results in duplicated efforts. We require a solution that enables users to discover indoor positioning systems. Users should be able to find the systems available to them and understand how to interact with them without requiring prior knowledge of these systems.

With this objective, we address RQ3. In addition, it also addresses RQ2.1, as the discovery of interfacing with systems helps with the access of data produced by such systems.

Objective 5 **Evaluate the feasibility of integrating interoperability and discovery in indoor positioning systems**
Finally, our last objective is to determine the feasibility of enabling the interoperability and discoverability of an IPS. With this objective, we want to go beyond theoretical interoperability and actually investigate whether this interoperability and discoverability can be achieved for current and future systems. In addition, we want to investigate whether we can combine discoverability and interoperability in a unified solution.

1.6 Methodology

In the previous sections, we presented the research questions that we will pursue to answer in this dissertation. This section will outline the general methodology we took to address these questions and validate our solution.

Our initial goal was to solve the main problem of fragmented indoor positioning systems due to the lack of a single standardisation, as clarified in Problem 1. Based on this problem, we determined that our first step would be to define a general representation of an indoor positioning system that covers different standardisation and current implementations. As these systems have many different implementations depending on the use case and used technologies, we investigated *hybrid* positioning systems beyond the state of the art (SOTA) of current indoor positioning systems (IPSs). The problems, research questions and objectives grew from this initial representation of a positioning system. While investigating the landscape and networking with researchers and developers, we also learned who would benefit from our solution.

Based on this preliminary research, we defined the actors and requirements of a hybrid positioning system. Using these requirements and actors, we then started with the design of a data-centric framework for representing the flow of positioning

systems and the data they produce (RQ1). Our focus with this framework was to design a data and processing representation of hybrid positioning systems rather than focusing on the development or end-user authoring of such systems. The framework was made open-source to enable feedback from peers. In addition, we used the framework from 2020 until 2025 in the course *Next Generation User Interfaces* and partly in the course *Web Technologies* to obtain feedback on its documentation and ease of use. Various proof-of-concept applications and systems were created as technical evaluations since 2019 to validate that our data and processing representation was generic enough to cover a wide range of use cases, technologies and implementations. These technical evaluations, detailed in Chapter 6, range from indoor positioning systems to augmented reality applications and form a basis for validating our data-centric framework. More details on the design and implementation methodology of this framework are given in Section 3.1.

At that point, we defined a framework that enables the processing and data representation of hybrid positioning systems. However, to solve Problem 1, we also needed to provide a method for enabling the seamless data exchange of such systems as defined in Objective 2. With the framework we designed in Chapter 3, we represented what a hybrid positioning system looks like. For answering RQ2, we We created an ontology for describing a wide range of positioning systems. Our methodology in this design was based on our representation of hybrid systems, ISO standards and commonly used terminologies. We validated this ontology through our previously designed framework and, in addition, also created several proof of concepts that use this ontology in Chapter 6. To ensure our ontology was extensible enough for future use cases, we expanded the ontology to other use cases and implementations beyond what we originally envisioned. The methodology of interoperable positioning systems is further detailed in Section 4.1.

Interoperability also entails that systems are capable of accessing data. In the context of positioning systems, we want to access data produced by such a system as stated in RQ2.1. This need for access to data stems from one of the requirements of interoperability in general, but also from the growing need of users to provide them with more control over their data. Our methodology for designing user-centric storage of data was based on our overarching research question for creating interoperable positioning systems along with current State of the Art (SOTA) on similar issues with users data.

Having user-centric storage of data not only provides users with the needed control over their data, but it also facilitates positioning systems to discover the data of users, as users explicitly have to grant access to this data. During the design of user-centric storage, we realised that this will not solve the fragmentation of indoor positioning systems due to users not being able to discover the existence of IPSs. We addressed RQ3 by proposing a solution that uses existing technologies within an IPS to discover the IPS itself. This way, we do not introduce new technologies,

but instead leverage the technologies that are being discovered to help with the discovery. To answer this research question, we based ourselves on the current SOTA on indoor positioning techniques as well as commonly used specifications.

While local discovery was our primary goal for discovering spatially bound indoor positioning systems, global discovery was still an open problem. Some of the indoor positioning systems use cases we have investigated in Section 2.2.1 presented the need for an alternative solution that does not rely solely on local discovery methods. This global discovery problem addresses the issue of how users can discover available indoor positioning systems in unfamiliar environments or when travelling to new locations. Our solution to this problem leverages our answers to the previous research questions to provide a distributed solution.

Finally, our methodology included the investigation and design of how to combine interoperability and discoverability into a solution that can be adopted by future indoor positioning systems.

1.7 Contributions

During our research on the interoperability and discoverability of positioning systems, various publications, tools, specifications, datasets, and other artefacts were created. One of our main contributions is the OpenHPS framework, which is used throughout all technical evaluations and prototypes of positioning systems. OpenHPS is the backbone of our research, providing both a standardised platform for testing and comparing different positioning technologies as well as a validation of the generalisation of positioning systems and their data.

Open-source Hybrid Positioning System OpenHPS

OpenHPS is an open-source framework for designing and developing hybrid positioning systems. It is developed in TypeScript and consists of over 25 modules that expand OpenHPS with positioning algorithms and sensor input. OpenHPS can run on smartphones, servers and web browsers. It is designed to support a wide range of use cases and is not tailored to one specific type of positioning system, platform or technology.

To aid in our research towards hybrid positioning systems and the data they produce, we developed a modular framework designed to be generic enough to create various positioning systems. We presented OpenHPS at the Indoor Positioning and Indoor Navigation (IPIN) conference in 2021, the Free and Open source Software Developers' European Meeting (FOSDEM) in 2022 and at the Belgian JavaScript Conference (BeJS) in 2023. OpenHPS is used for all subsequent research and prototypes and was used to validate our research questions RQ1, RQ1.1 and RQ1.2.

OpenHPS, as a framework, comprises a conceptual design representation of a hybrid positioning system, the data it produces and the implementation of this design. With this contribution, we provided a standard data structure that could serve as the basis for a future generic standardisation of hybrid or indoor positioning systems. Our framework implementation is aimed towards making the representation tangible and usable, in order to validate its integration into existing and future deployments.

Relevant Publications:

- Maxim Van de Wynckel and Beat Signer. OpenHPS: An Open Source Hybrid Positioning System. Technical Report WISE-2020-01, Vrije Universiteit Brussel, 2020. doi: 10.48550/ARXIV.2101.05198

- Maxim Van de Wynckel and Beat Signer. Indoor Positioning Using the OpenHPS Framework. In *Proceedings of the 11th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2021)*, 2021. doi: 10.1109/IPIN51156.2021.9662569

- Maxim Van de Wynckel and Beat Signer. OpenHPS: A Modular Framework to Facilitate the Development of FAIR Positioning Systems. *Journal of Open Source Software*, 10(110):8113, June 2025. doi: 10.21105/joss.08113

- **Dataset:** Maxim Van de Wynckel and Beat Signer. OpenHPS: Single Floor Fingerprinting and Trajectory Dataset, May 2021. doi: 10.5281/zenodo.4744380

- **Dataset:** Benjamin Vermunicht, Maxim Van de Wynckel, and Beat Signer. 802.11 Managemement Frames From a Public Location, June 2023. doi: 10.5281/zenodo.8003771
  Used for analysing the input and output data in @openhps/rf

- **Dataset:** Nathan Hoebeke, Maxim Van de Wynckel, and Beat Signer. Object Tracking on a Monopoly Game Board, June 2023. doi: 10.5281/zenodo.7990434
  Used for analysing the input and output data in @openhps/video

- **Dataset:** Maxim Van de Wynckel. Garage Positioning Dataset, 2025. doi: 10.34740/KAGGLE/DS/6654647

- **Dataset:** Maxim Van de Wynckel and Beat Signer. Sphero Dead Reckoning and CV Tracking Dataset, 2025. doi: 10.34740/KAGGLE/DS/6760212

Positioning System Ontology POSO

The Positioning System Ontology (POSO) describes positioning systems, their algorithms and the data they produce using RDF. POSO is designed to facilitate interoperability between positioning systems by defining a common vocabulary for describing the data and processing pipeline of this data.

We presented POSO at the International Semantic Web Conference (ISWC) in 2022 and its implementation within OpenHPS for serialising and deserialising data produced and consumed by positioning systems. In 2024, we extended POSO with our fiducial marker ontology called *FidMark*, which was presented at the Extended Semantic Web Conference (ESWC). With this extension, we demonstrated the importance and possibility of creating interoperability in augmented reality environments while simultaneously evaluating the extendability of the POSO ontology.

With POSO, we provide an extensible solution to answer research question RQ2. POSO builds upon the conceptual representation of a positioning system that we defined with OpenHPS. Using the data structure and pipeline we defined for OpenHPS, together with existing terminologies and standardisations, we provide an ontology that describes this representation.

Relevant Publications:

- Maxim Van de Wynckel and Beat Signer. POSO: A Generic Positioning System Ontology. In *Proceedings of the 21st International Semantic Web Conference (ISWC 2022)*, Virtual conference, 2022. doi: 10.1007/978-3-031-19433-7_14
- Maxim Van de Wynckel, Isaac Valadez, and Beat Signer. FidMark: A Fiducial Marker Ontology for Semantically Describing Visual Markers. In *Proceedings of The Semantic Web (ESWC 2024)*, 2024. doi: 10.1007/978-3-031-60635-9_14
- Maxim Van de Wynckel and Beat Signer. Discoverable and Interoperable Augmented Reality Environments Through Solid Pods. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL https://ceur-ws.org/Vol-3947/short2.pdf

RDF Object-Document Mapping

We investigated and designed a solution for automatically mapping objects to RDF. Our implementation was designed as a module for OpenHPS (`@openhps/rdf`). This document mapper is flexible and can be configured to perform RDF mappings without changing the existing object-oriented design.

To aid in designing interoperable positioning systems, we investigated how to seamlessly map positioning systems and their data from the OpenHPS framework to Resource Description Framework (RDF) using the POSO ontology. This contribution helped us validate the POSO ontology's genericity by mapping it to concepts of our generic OpenHPS framework.

Our solution was to design an object-document mapping tool that allows serialisation and deserialisation of the existing OpenHPS classes and objects to RDF. Existing solutions often require the manipulation of classes to be mapped to RDF. In contrast to this existing work, our solution does not require any modifications to the classes themselves. This contribution serves as an implementation that bridges the gap between our ontology and the practical implementation of OpenHPS. However, it also provides novel solutions to help bridging this gap.

While our implementation was built on top of OpenHPS, the offered solution for extensible serialisation to RDF and to query data by providing MongoDB queries is generic enough to be utilised in other use cases.

User-centric Positioning System Storage Using Solid

Using *POSO*, *OpenHPS* and our *RDF object-document mapper*, we created a Solid-based positioning system that describes a positioning system and the data it contains. This data is stored in personal data vaults managed by the user. We treat data such as a *position* or *orientation* as an observable property that is linked to a user's Web Identifier. Based on access rights, applications can *observe* the properties of a user that are stored within Solid.

Our initial solution for decentralising location data was presented at IPIN 2022. Based on the feedback we received for this initial version, we modified our solution by optimising the structuring of the data within a personal data vault such as a Solid Pod. We presented other use cases of decentralised location data at the Solid Symposium, where we proposed two methods for utilising Solid as a communication broker for data modifications.

This contribution serves as a conceptual architecture that uses personal data vaults for enabling user-centric storage of data obtained or used by positioning systems.

However, to validate that this architecture is feasible, we have implemented it on top of the Solid project, where we defined new concepts for enabling this architecture within Solid.

Relevant Publications:

- Maxim Van de Wynckel and Beat Signer. A Solid-based Architecture for Decentralised Interoperable Location Data. In *Proceedings of the 12th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2022), CEUR Workshop Proceedings*, volume 3248, 2022. URL `https://ceur-ws.org/Vol-3248/paper11.pdf`
- Maxim Van de Wynckel and Beat Signer. Discoverable and Interoperable Augmented Reality Environments Through Solid Pods. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL `https://ceur-ws.org/Vol-3947/short2.pdf`
- Yoshi Malaise, Maxim Van de Wynckel, and Beat Signer. Towards Distributed Intelligent Tutoring Systems Based on User-owned Progress and Performance Data. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL `https://ceur-ws.org/Vol-3947/short3.pdf` *This paper was used to investigate a different approach to use Solid as a communication method for triggering actions*

Semantic Beacons SemBeacon

The *SemBeacon* specification is a semantic beacon solution for advertising semantic data about geospatial environments and physical things. It is based on the Bluetooth Low Energy specifications, AltBeacon and Eddystone-URL, and is also backwards compatible with libraries or scanners that already scan for these existing specifications. The SemBeacon specification includes an ontology alignment with POSO to ensure seamless integration with existing positioning systems and data processing pipelines.

SemBeacon is our solution for enabling the local discovery of positioning systems. While SemBeacon is primarily an implementation for discovering services and physical objects, its design is problem-oriented by reusing the technologies which are being discovered to aid in the discovery. This way, we created a solution that preserves the interoperability as much as possible.

We first demonstrated the use of SemBeacon to discover and describe physical things at the international Internet of Things (IoT) conference in 2023. We then continued the development of the specification so it could be used in subsequent research. We developed and published an Android application and Arduino library to bring SemBeacon to as many users as possible. In addition, we extended a

popular Android Bluetooth scanning library to scan for and emulate SemBeacon to promote its adoption among developers.

Finally, using SemBeacon, POSO and OpenHPS, we have explored the ability to combine these three projects to create discoverable digital twins of environments, positioning systems and reference spaces. Using FidMark together with our other research, we concluded our research with two papers towards discoverable and interoperable augmented reality environments within Solid Pods, which we presented as a poster at the 2nd Solid Symposium in 2024. We also presented the use of SemBeacons for discovering indoor environments at FOSDEM 2025.

Relevant Publications:

- Maxim Van de Wynckel and Beat Signer. SemBeacon: A Semantic Proximity Beacon Solution for Discovering and Detecting the Position of Physical Things. In *Proceedings of the 13th International Conference on the Internet of Things (IoT 2023)*, 2023. doi: 10.1145/3627050.3627060

- Maxim Van de Wynckel and Beat Signer. Discoverable and Interoperable Augmented Reality Environments Through Solid Pods. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL https://ceur-ws.org/Vol-3947/short2.pdf

Linked Data Hash Tables

We designed a new specification called Linked Data Hash Tables (LDHT) that utilises linked data to create an Distributed Hash Table (DHT). It enables a global distributed collection of data. Data retrieval uses the Semantic Web to traverse the nodes in the distributed network. LDHT is a generic global discovery method that we use to distribute knowledge about indoor positioning systems.

SemBeacon aids in the *local* discovery of positioning systems by leveraging the signal propagation properties of Bluetooth Low Energy to receive information about positioning systems within proximity of these beacons. However, this requires additional infrastructure for broadcasting information. Linked Data Hash Tables is a specification that leverages the Linked Data Notifications protocol to create a distributed collection of URIs linked to geographical boundaries. We utilise this to provide a global discovery of positioning systems.

At the time of writing, the work on LDHT has not yet been published in an academic paper. However, the design is influenced by our use of Solid as a communication broker with *actions* that can be sent to each Pod [42, 44].

Integrated Solution

Our integrated solution combines SemBeacon, POSO, our decentralisation with Solid and linked data hash tables into an architecture for designing interoperable and discoverable indoor positioning systems.

Using all our combined contributions, we designed an architectural system for building future interoperable and discoverable indoor positioning systems. This integrated solution uses Solid personal data vaults to store properties of location, orientation and other data. This data is described using the POSO ontology and constructed by the OpenHPS framework. Our integrated solution represents a conceptual blueprint for designing *Interoperable and Discoverable Indoor Positioning Systems*. We utilise our implementations to detail our vision of this blueprint, but the conceptual architecture can be applied using different discovery methods and data storage providers.

Other than the contributions listed in this section, we also contributed several smaller software utilities that were separated from our SemBeacon contribution to facilitate future researchers in their research efforts involving linked data. These include a standalone server for shortening URIs in a way that supports SPARQL engines to access the data and a standalone linked data HTTP proxy server that aims to provide a CORS-compliant proxy for accessing the Semantic Web. These software utilities will not be further discussed but are available on GitHub[3,4].

Relevant Publications:

- Maxim Van de Wynckel and Beat Signer. Survey on the Privacy and Transparency of Location Data, May 2025. doi: `10.5281/zenodo.15564050`
  Survey used to position the problem and assess the motivation for user-centric storage.

Finally, in Appendix E, we provide the results of a survey we conducted in 2025 to assess users' perspectives towards the privacy and transparency of location data. With this survey, we wanted to determine how our integrated solution can be improved in the future. Part of the aggregated results of this survey were discussed in our introduction. Note that in the subsequent chapters, we have reused some content from our previously published papers that have been mentioned under the contributions in this section, specifications, blog posts and documentation from `*.openhps.org` and `sembeacon.org`.

[3] `https://github.com/SemBeacon/shortener`
[4] `https://github.com/SemBeacon/proxy`

You can't really know where you are going
until you know where you have been.

– Maya Angelou

Chapter 2

Background and Related Work

Designing an interoperable and discoverable positioning system or service is a challenging problem due to the broad domain it has to cover. A positioning system is a collection of various technologies and algorithms designed to work in pre-specified conditions, such as specific environments or using calibrated hardware. Creating *interoperability* for these positioning systems would first require the seamless exchange of the data that these systems consume and produce and the semantic description of the processing steps of this data to enable reasoning on the produced data and its quality. Furthermore, while we can often assume that positioning systems aim to determine a location on Earth, this is not always the case. Often, such a system only aims to provide a location for the environment it is responsible for, which can be a single floor of a building or even something as small as a physical game board.

The *discovery* of positioning systems is the ability to *find* or *learn* about the existence of positioning systems in an environment. This can involve discovering the technologies and techniques these systems use, their capabilities, data requirements and how they can be accessed and utilised. However, this discovery also encompasses the discovery of data belonging to a particular person or object that is being tracked by such a system, so that seamless data exchange can happen between different systems.

In this chapter, we delve deeper into the background of positioning systems, the interoperability between these systems, how we can achieve the discovery of such systems and the entities whose position is being determined. We focus on existing standardisations, research efforts and challenges that closely relate to the research questions we are trying to answer. All of the background in this chapter is used throughout the dissertation, either to compare the state of the art (SOTA), or to investigate the requirements of our individual solutions.

2.1 Basic Terminology

Before we discuss the concepts of interoperable and discoverable *indoor* positioning systems, we need to establish a solid understanding of some basic terminology used throughout the dissertation. In this section, we primarily focus on the different available positioning systems, the fundamental data that a positioning system should be able to provide, and the naming conventions of this data. In later sections, we delve deeper into system- and algorithm-specific terminology and their implications on indoor positioning systems.

2.1.1 Positioning Systems and Services

Whenever we want to determine the position or orientation of a person or object, we require sensors and software to retrieve this information. A positioning system is a system or mechanism that can determine the position of one or multiple objects based on some sensor data. Multiple positioning systems might track the same object either individually or simultaneously. These multiple systems can work independently from each other or combine information from other systems to provide an output.

The ISO 19116:2019 standard [46], which defines *geographic* positioning services, distinguishes between five main types of positioning systems:

- **Satellite positioning system**: A positioning system using satellites in geo-stationary orbit around the Earth. These types of positioning systems are also referred to as Global Navigation Satellite Systems (GNSS). Examples include the GPS, Galileo or GLONASS [47].

- **Integrated positioning system**: An integrated or hybrid positioning system (as detailed in Definition 4) can be used outdoors, indoors or in any other space and contains a combination of two or more technologies. It describes any system that combines multiple technologies, regardless of whether they also fit in one of the other positioning systems within the ISO 19116:2019 standard. Assisted GPS (A-GPS) [48] is an integrated positioning system that combines GPS with other techniques such as dead reckoning or cell tower, making this a combination of a satellite positioning system and an inertial positioning system.

- **Optical positioning system**: A positioning system that uses optical sensors to determine a position. This includes positioning systems where objects are tracked externally (e.g., Multi-Target Multi-Camera Tracking [49]), systems where the tracked object is the optical sensor observing the environment (e.g., Visual Simultaneous Localisation and Mapping [50]) or fiducial marker trackers [51, 52].

- **Inertial positioning system**: An inertial positioning system calculates the position based on its movement, measured through inertial measurement sensors and an initial reference point [53]. The basic concept of these types of positioning systems is further detailed in Section 2.3.7.
- **Linear positioning system**: A linear positioning system calculates its position along a linear axis, such as an odometer [46] tracking the linear movement.

Since the ISO 19116:2019 standard only considers geographical positioning systems, we also consider the terminology of indoor positioning systems as an additional variant, based on the *indoor navigation system* concept defined in ISO 17438-4:2019 [21]. An indoor positioning system covers all systems and techniques that are deployed indoors as opposed to outdoor positioning, where often satellite positioning is used [54]. Indoor positioning systems are tailored to the building in which they are deployed and the use case they tend to solve.

This type of positioning system can be projected to a geographical coordinate, but in use cases where the positioning is only required relative to the indoor environment — this is not always the case. In such systems, Cartesian coordinates relative to the building or floor determine the position rather than projecting these coordinates to geographical coordinates. Furthermore, while indoor environments are often fixed, this is not always the case, as seen on cruise ships that contain a complex indoor environment but should not always be mapped to geographical coordinates.

In general, our solution and research in this dissertation is based on a wide variety of positioning systems, technologies, algorithms and use cases. However, with our goal of designing interoperable positioning systems, we focus on the background information for indoor positioning systems, as these are the types of systems that do not provide standardisation and often utilise proprietary databases owned by a building owner. We consider an indoor positioning system as a type of hybrid or integrated positioning system due to these types of systems often not relying on a single technology.

Hybrid Positioning Systems

Every technique used to determine a position has its advantages and disadvantages. Some techniques require significant infrastructure changes, while others require a lot of processing power. Moreover, not every technique works perfectly in all environmental conditions, preventing the use of a single technology to determine a position. To solve this issue, hybrid positioning combines multiple systems or sensors to provide more reliable output. The *hybrid positioning system* terminology covers a wide range of use cases and therefore offers a good reason for further research to identify a common data representation of such systems.

One of the common tactics of hybrid positioning systems is the fusion of sensor data to improve the overall accuracy and reliability of the positioning system. The use of sensor fusion mitigates the limitations of individual sensors and enhances the overall system performance.

Another hybrid technique that is similar to sensor fusion is data reasoning, also called inference-based positioning. This technique leverages contextual information to precisely determine a sensor's output. This method not only utilises sensor data but also combines it with other positioning techniques and high-level data to enhance the accuracy and reliability of the final output. By considering various factors such as environmental conditions, historical data, and signal strength, inference-based positioning can provide a more comprehensive and robust positioning solution.

Figure 2.1: Employee using a badge to enter a secured area

A basic example of data reasoning for position estimation is an employee who is tracked inside a building using a non-trivial set of positioning techniques. The employee has a personal badge to gain access to closed-off areas within the company. Other than traditional tracking methods which aim to determine a general location in a building, when the employee uses their badge to gain access to a specific room, their location can be more accurately inferred to be at the location where the badge scanner is located. As shown in Figure 2.1, these badges are often personal (e.g., a staff card) and can therefore be used to assume that at the time when they are used, the user is entering an environment.

Various research studies concerning the fusion of sensor data to predict a more accurate position exist. SignalSLAM [55] represents an example of a hybrid

system that uses signals of various positioning methods such as GPS, Wi-Fi and Bluetooth to map the surroundings. Chen et al. [56] have shown how a smartphone can combine sensor data of Wi-Fi access point positioning and pedestrian dead reckoning. This combination of dead reckoning with another positioning method is a common combination used by many hybrid systems.

In the research by Bekkelien and Deriaz [57], a framework called Global Positioning Module (GPM) had been presented for in- and outdoor positioning. GPM provides a uniform interface to different positions *providers*. These providers are fused in a *kernel* that selects the position based on provided *criteria* (e.g., precision, accuracy or detection probability). Their approach offers a clear methodology on how these criteria can contribute to the selection or fusion of different technologies. However, the position providers and kernels are implemented on a high level of abstraction, providing no room for developers to choose different algorithms or fusion techniques.

Ficco and Russo [58] presented a technology-independent hybrid positioning middleware called HyLocSys. Position *estimators*, representing different technologies, provide positions when a user performs a *pull* of their current position. Sensor fusion combines these estimated positions into a final response. The middleware is based on the JSR-179 specification [59], which is a framework for mobile devices to obtain a location. This framework enables pull requests for a location to accept criteria such as the preferred response time and expected accuracy. Other than many frameworks that only provide geographical positions, HyLocSys provides geometric, symbolic, as well as hybrid location models. Symbolic locations represent abstract places such as buildings, floors and rooms that are positioned relative to each other. A hybrid location can convert this symbolic location to a geometric position. Note that the paper does not discuss positioning technologies such as dead reckoning or SLAM that require periodic updates to keep an up-to-date position.

Scholl et al. [60] propose a system that uses a Light Detection and Ranging (LiDAR) scanner to determine the fingerprinting position. Using this technique, a more precise positioning technique is used to aid in the calibration of another technique. The Robot Operating System (ROS) [61] is a structured communication layer that can be used to create autonomous robots. It focuses on the integration of various robotics aspects such as positioning, computing and hardware interfacing. ROS provides the concept of peer-connected nodes that perform computational tasks. These nodes represent interchangeable software modules that help to build a pipeline from sensory data to an output action. For positions and orientations, ROS uses the *pose* concept, which contains both the position and orientation of a robot. In full body tracking of users, this *pose* terminology is also used [62] as it is also considered a non-technical terminology for indicating the posture of a person.

With hybrid positioning covering a wide variety of use cases and technologies, it offers a stable foundation to generalise the types of positioning systems that

could be implemented. In our dissertation, we first focused on generalising hybrid positioning systems with the development of our OpenHPS framework.

Location-based Services

A location-based service (LBS) is an abstraction of the underlying positioning system with its implemented techniques that offer third-party applications a *service* to retrieve the position of a person or object [63]. An example of a LBS is the Geolocation API implemented in most modern web browsers [64]. This API offers an application interface to retrieve a position or watch for changes in the position. However, depending on the available hardware, the underlying technology varies from Wi-Fi positioning to GPS or even IP-address-based location estimation. However, developers can request a high-accuracy result or maximum cache age if the hardware permits this. The resulting positions of the Geolocation API are geographical coordinates complying with the WGS-84 standard [65].

JSR-179 and the improved JSR-293 [59] specifications are Java 2 Micro Edition (J2ME) modules that provide developers with an API to obtain the location and orientation of a mobile device. Included in the API is a storage interface for landmarks, which can be used by RF-based positioning techniques to store fixed landmarks for positioning techniques such as multilateration. The specification represents locations as timestamped coordinates with an orientation, accuracy, speed and information about the used positioning method [66]. When requesting a location, criteria such as the desired accuracy, power consumption and response timeout can be provided.

With this dissertation, we want to focus more on the interoperability of the underlying technologies rather than abstractions such as LBSs. While we still consider location-based services when providing a solution to our research questions, we do not aim to provide interoperability between these high-level services.

However, as we will point out at the end of this dissertation, future work should also specifically focus on the interoperability of location-based services. Providing interoperable indoor positioning systems is one key challenge that we address in this dissertation, but without an abstraction layer that manages and controls this, the full potential of an interoperable system cannot be realised.

2.1.2 Absolute and Relative Positions

Multiple definitions exist to indicate where a spatial object is located. Positioning systems distinguish between *relative* and *absolute* positions [2, 54]. An absolute position represents a fixed position in a specified *space* while relative positions indicate the position relative to another object or reference space. Absolute positions indicate a position on a specified coordinate system, such as latitude and longitude in GPS. In contrast, relative positions are more dynamic and can change depending

on the reference point. Hybrid positioning systems can utilise both absolute and relative positioning techniques to enhance the accuracy and reliability of the overall system. Figure 2.2 illustrates the difference between an absolute and relative position. A boat may have an absolute position at a specific coordinate, but can also have a relative position to a lighthouse.

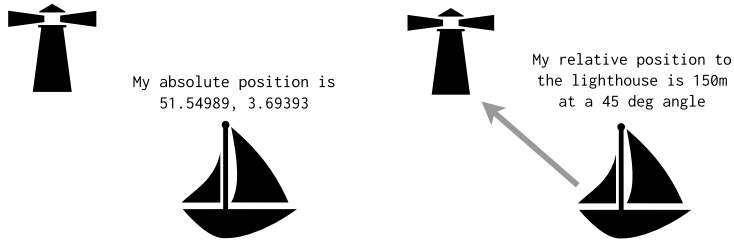


Figure 2.2: Absolute and relative position of a boat

The position terminology is often used as opposed to *location* or *pose*. Pose is a term that is often used when defining a position and orientation in a three-dimensional space and is often used in robotics [67] or when describing the movement of a person [68]. A pose is a combination of the position and orientation of an object. In real-world applications such as the Robot Operating System (ROS) [61], it indicates a position and orientation within 3D space. Not every positioning system might operate within three dimensions, so the *pose* terminology might not be appropriate as a generic terminology. Location is described by the English Oxford Dictionary as *"a place where something happens or exists; the position of something"*. Since a *place* denotes a larger area where something happens or exists, we decided on the position terminology.

2.1.3 Coordinate Systems

A coordinate reference frame (CRF), coordinate reference system (CRS) or other defined as *"a frame of reference"* is a standard way of specifying the coordinates of points in space. It provides a set of axes that define the position of a point or an object, which can be used to specify its location. Different applications may require different coordinate reference frames depending on the desired level of accuracy and the specific needs of the system. For example, in the field of robotics, a Cartesian coordinate system is often used to define the position of a robot in a 2D or 3D space. Other applications, such as satellite position systems, may use spherical coordinate systems to represent positions on the Earth's surface. The choice of coordinate reference frame depends on the specific requirements of the application and the nature of the spatial object being measured.

Another terminology that is often used to describe a position within a certain frame of reference is a *local- and global reference space*. In a local reference space, the

positioning system may define its own coordinate system, its own origin and its own rotation without being earth-bound. Positions within a local reference frame can optionally be mapped to a global reference space which maps the local coordinates to an existing coordinate reference frame. An example of such a reference space is a system that can track relative movements but has no knowledge about its absolute position. A virtual reality (VR) headset may be able to track its movement and orientation relative to the start of the VR session but often does not have knowledge about its absolute position.

In our efforts to design interoperable positioning systems, we must consider the different coordinate systems and reference spaces that may be used in various applications. Whenever expressing location data, we should provide context about the coordinate system used to express this location.

2.1.4 Orientation

In the context of positioning systems, orientation refers to the direction in which an object is facing or aligned. Depending on the type of system, this orientation can be expressed with a three-dimensional vector or a simple angle.

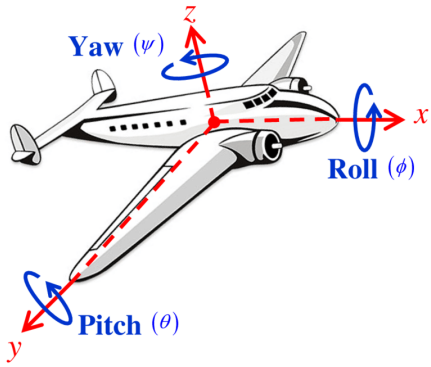


Figure 2.3: Visual explanation of Euler angles [69]

Orientation is an important aspect of a positioning system. It not only offers the final state of direction after a rotation of an object or person, but is also required by many positioning algorithms to determine a position. In a geographical context, the terminology *bearing*, *heading*, *course* or *azimuth* is used as a one-dimensional value [70]. To offer a more generic terminology of a two- or three-dimensional position, mathematical concepts can be used to describe an orientation.

The commonly used mathematical definitions of orientation are *Euler Angles*, *Axis Angles*, *Quaternions* and *rotation matrices* [71]. Each mathematical definition has its advantages for a positioning system. Euler angles offer a well-known semantic

description of a 3D rotation using only three values as shown in Figure 2.3, while still allowing the single use of *yaw* for expressing the heading in a 2D scenario. In robotics, quaternions are chosen since they avoid gimbal locks, as well as for their analytic properties. Rotation matrices are commonly used to easily combine a rotation matrix with a position matrix that contains the scale and translation of an object within a space.

2.1.5 Velocity and Acceleration

Velocity consists of angular and linear velocity. Linear velocity is the velocity along one or more axes relative to the current orientation while angular velocity is the velocity around the position. When working with this type of data, the timestamp of when this information was created is crucial to accurately compute a location and orientation.

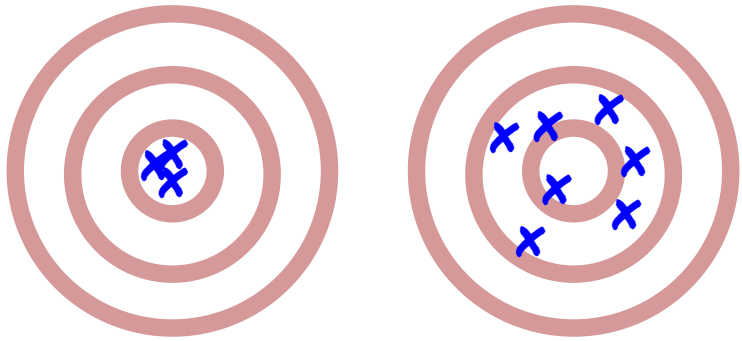
Acceleration on the other hand is the rate of change of velocity, both linear and angular. It measures how quickly the velocity of an object is changing over time. Like velocity, acceleration can be in different directions and is also dependent on the timestamp of when the data was collected.

Active positioning systems make use of an object's velocity to determine a position and orientation based on its momentum. This procedure, called dead reckoning, uses an entity's last known location together with its angular and linear velocity to determine the new position and orientation at a later timestamp [72].

2.1.6 Accuracy and Precision

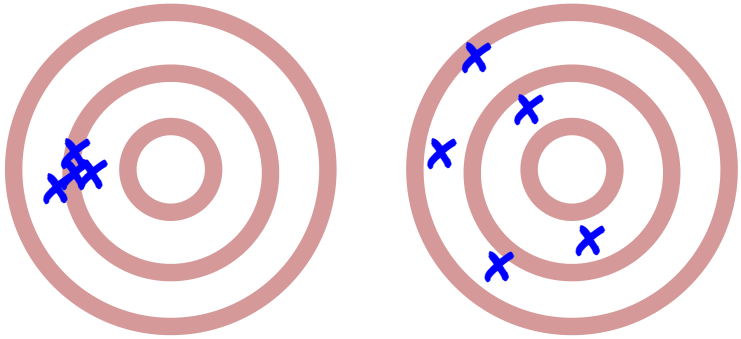
Accuracy and precision are vital concepts in hybrid positioning systems as they can be used to determine the relevance of a particular algorithm or system. Accuracy refers to how close a measured value is to the true value, while precision relates to the consistency and repeatability of the measurements. When computing information in a positioning system, accuracy describes how closely the calculated position matches the actual ground-truth position, while precision indicates the level of detail or granularity in the measurements.

When we are developing positioning systems, the accuracy indicates how close the computed position is to the actual position of the object or person that is being tracked. Precision, on the other hand, is related to the level of detail in the location measurements and how consistently these location measurements can be reproduced over time. Figure 2.4 illustrates the difference between accuracy and precision using four examples. Mathematically, accuracy can be defined as the absolute value of the difference between the measured value and the true value, while precision refers to the scatter of measured values around the mean. In a positioning system, high accuracy with high precision is the desired outcome, as it



(a) High accuracy with high precision

(b) High accuracy with low precision



(c) Low accuracy with high precision

(d) Low accuracy with low precision

Figure 2.4: Accuracy and precision

indicates that the calculated positions are not only close to the actual positions but also consistently reproducible.

Interoperable positioning systems that exchange data should give assurances on the data's accuracy and precision to ensure that the information being shared is reliable and consistent across different systems. This is important for applications such as collaborative robotics, autonomous vehicles, and augmented reality, where multiple devices and sensors need to work together seamlessly. Part of our research efforts is to ensure that this knowledge can be exchanged.

2.2 Indoor Positioning Systems

Outdoors, we can rely on satellite positioning systems such as GPS to help determine a position; inside a building, we cannot rely on this technique due to the obstacles between the satellites and the receiver within a building. Hybrid positioning is often used within these environments to handle this limitation. In indoor environments,

positioning systems face additional challenges such as multi-path effects [73], signal interference [74] or even blind spots within an environment where users cannot be tracked. These challenges require the use of different positioning techniques that are specifically tailored for indoor environments.

These positioning techniques may include Wi-Fi-based positioning, Bluetooth Low Energy (BLE) beacons, ultrasound positioning, infrared positioning, and computer vision-based approaches. Each of these techniques has its strengths and limitations, making it essential to choose the most suitable technique based on the specific requirements of the indoor positioning application and the building environment.

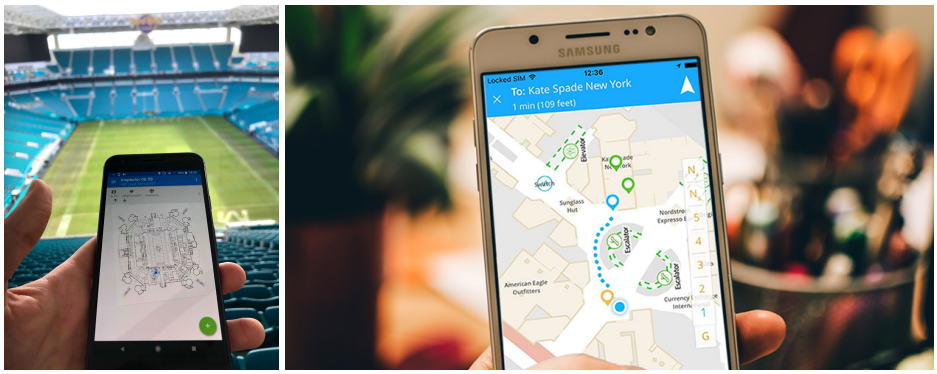


Figure 2.6: Two indoor positioning systems developed by IndoorAtlas[1]

Figure 2.6 showcases two indoor positioning systems developed by IndoorAtlas, a company that develops and deploys indoor positioning and navigation systems. IndoorAtlas provides customers with a platform as a service with a well-established Software Development Kit (SDK) for combining Wi-Fi, GPS, Bluetooth beacons, dead reckoning and even geomagnetic positioning [75, 76]. While the latter method is less ideal in steel-reinforced buildings [77], it still offers a useful addition for creating an indoor positioning system where geomagnetic positioning might be combined with other positioning methods.

GoodMaps[2] is a company that provides indoor mapping services for buildings. They perform a visual LiDAR scan of a building after which they create a 2D floor-plan. During the online phase, the features detected through this scan are used to determine a user's location.

If we look at more open platforms and frameworks, LearnLoc [77] is a smartphone positioning framework that uses the k-nearest neighbours fingerprint algorithm in combination with various sensor data available on a smartphone to provide power-efficient indoor positioning. The consideration of power efficiency is a common

[1] https://www.indooratlas.com
[2] https://goodmaps.com

requirement in mobile positioning systems, which this platform tries to address. Other than achieving the most accurate position, these types of location-based services (LBSs) use sensor fusion to prevent the continuous use of precise sensors such as cameras or GPS.

One of the things most indoor positioning systems have in common is the need to either perform a major calibration step by scanning or collecting fingerprints, or to install hardware within a building. Both of these approaches have their pros and cons, but are generally considered costly or sensitive to environmental changes. When these systems additionally require their own tailored application, they become even more expensive and difficult to set up. Interoperable indoor positioning systems can help to alleviate these challenges by providing standardised approaches that can work across different platforms and devices, reducing costs and simplifying the setup process for indoor positioning systems.

2.2.1 Use Cases

Indoor positioning systems are not free or effortless to deploy. Indoor navigation applications require the development of a custom application, the mapping of the indoor environment and the purchase and installation of specialised hardware such as beacons or RFID tags [78]. Therefore, indoor positioning systems are often not primarily deployed to simply enhance user experience but are rather used to optimise business processes. However, once an indoor positioning system is deployed, it can offer a wide variety of other use cases, including the enhancement of user experience.

Figure 2.7 illustrates the main domains and building types where indoor positioning systems can or could be used. In this section, we further detail these use cases to provide additional context on how an indoor positioning system is used. By understanding these use cases, we can determine the requirements of such systems and how they are deployed.

Airports

Airports are large and often unfamiliar to travellers. Some airports have multiple terminals that take a significant amount of time to reach. Indoor positioning systems can help passengers navigate through the airport efficiently. A personal navigation indoor navigation system helps passengers locate their check-in desk, terminal and gate within the airport. It can also help them locate the location of their luggage when reaching their destination.

Security systems at an airport already leverage surveillance cameras to track specific passengers. Currently, there is no transparency towards the users regarding how they are being tracked, and their use is limited to security purposes. However,

Figure 2.7: Indoor positioning system use cases

with an indoor positioning system in place, airports could potentially request user consent to let staff track a specific passenger.

Museums

Museums are a popular use case for indoor positioning systems because *enhancing user experience* is part of the business goals of museums. Visitors can use their smartphones to access interactive maps that guide them through the exhibits, providing additional location-based information and context about each artefact [79]. Indoor positioning can also help museum staff track visitor flow, gather data on popular exhibits, and improve crowd management.

Museums also aim to engage visitors by creating interactive exhibits. Augmented reality is a technology that can help with this interaction. However, these AR applications also require an indoor positioning system to correctly augment information on physical exhibits [80].

Conferences and Events

Conferences are often crowded events with multiple stands, multiple rooms for presentations, and various activities happening simultaneously. Indoor positioning systems are already being deployed in large conference halls to facilitate navigation. However, these systems are often temporary deployments that require a single-use application.

In addition to simple navigation purposes, person-to-person navigation is a type of navigation where users have to navigate to another person. In a conference or meetup, this type of navigation can be used to find specific speakers or participants within the venue. This can enhance networking opportunities and streamline the process of finding and connecting with individuals in a crowded event space.

For organisers, indoor tracking can provide insightful analytics on popular stands, busy walkways, and peak times during the event. This data can help better organise future conferences and meetups to improve the attendees' experience.

Campus

On a university campus, indoor positioning systems can be used to assist students and visitors in navigating the complex network of buildings and facilities. A campus often spans multiple buildings, meaning students need to go from one building to the next.

Next to user experience, an indoor positioning system can help to determine if rooms are available or use attendance information in combination with other contexts (e.g., air quality sensors) to automatically determine if different rooms should be assigned to courses with many attendees.

Transportation

In outdoor environments, transportation can easily be tracked using GPS. While it is not necessary for an *indoor* environment, underground transportation also requires tracking, which is not feasible using GPS signals. Furthermore, transportation hubs, such as stations, are often large and complex, making it difficult for passengers to navigate efficiently.

Figure 2.8 shows the floor plan of Tokyo Station, Japan. A station with 28 platforms, multiple entrances and multiple floors to get to the platform. For a visitor, it is already a challenge to navigate to the destination, but the complex environment also makes it difficult for a person to know exactly where they are and which entrance they took.

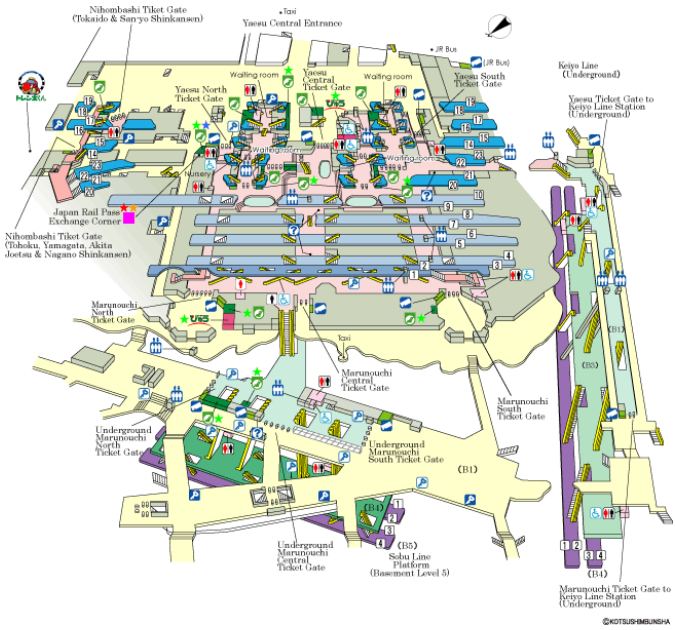


Figure 2.8: Tokyo station multi-level floor plan (source Kotsu Shimbunsha)

Hospitals

Hospitals are complex environments that are often also visited by people with accessibility issues. These environments often provide a *route* that can be followed to their destination. However, for people with a visual impairment, reading these numbers or following these colours is difficult. Furthermore, these routes are often not optimised for efficiency, causing patients to sometimes get lost or have delays in reaching their destination. Indoor positioning systems can provide turn-by-turn navigation for patients, visitors, and staff within a hospital, improving efficiency and reducing stress [81].

Apart from improving visitor/patient experience, hospitals also have movable assets such as EKG machines that are moved from department to department depending on where they are needed most [82]. Asset tracking can help manage where these assets are located, to ensure that these portable devices are not lost or misplaced. This can save time for staff who would otherwise have to search for these devices, as well as improve patient care by ensuring that necessary equipment is readily available when needed.

Retail Stores and Malls

Out of all use cases, retail stores and large malls are currently one of the main businesses that implement indoor positioning or indoor tracking solutions [83, 84]. In warehouses, use cases range from asset tracking to automatic order picking

using robots. For retail stores, it can be used to determine how long customers spend in certain areas or to provide location-based advertisements on their phones. Figure 2.9 showcases a robot[3] that helps employees perform order picking. These robots need to navigate within a warehouse to the correct locations where these orders should be picked.

Figure 2.9: Assisted order picking robot from Zeal Robotics

From a user's perspective, some of these use cases are not necessarily seen as beneficial, as they may invade their privacy or feel like an intrusion into their personal space. However, from a business perspective, the data collected from these systems can provide valuable insights into customer behaviour and preferences, which can be utilised to improve marketing strategies and optimise the layout of the store for a better shopping experience.

In large malls or even on cruise ships with multiple shops, indoor positioning systems can help visitors navigate through the complex layout of the space, find specific stores or amenities, and allow them to receive personalised recommendations based on their preferences. This can enhance the overall shopping experience for visitors, leading to increased customer satisfaction and potentially higher revenue.

[3] https://www.zealrobotics.com

Office Environments

After the COVID-19 pandemic, more companies started to investigate smart co-working spaces. These co-working spaces are flexible workplaces where employees are not assigned a fixed desk but rather expected to work at a *flex desk*. The idea behind this concept is that employees do not always work on-premises, and therefore, companies should not be constrained by fixed desk assignments.

Managing co-working spaces requires a robust *registration* system to determine which rooms or desks are available. Meanwhile, since employees no longer have a fixed desk, it becomes more difficult to have face-to-face meetings with co-workers. An indoor positioning system can help with person-to-person navigation as well as the automatic detection of which rooms or desks are available.

Similar to hospital environments, office environments can also benefit from asset or resource tracking. Portable equipment that moves from one office to another should be easily tracked, especially in shared workspaces. Flexible workspaces also entail that employees are not familiar with the nearby equipment, such as printers. Resource tracking can help them find the location of the closest printer.

Figure 2.10: Sony Nimway wall panel

Figure 2.10 showcases a Sony Nimway [85] wall panel. Nimway is a smart office solution that handles employee tracking, resource allocation, last-minute bookings and office navigation. It offers executives and office managers insights into employee attendance, desk utilisation and occupancy.

2.2.2 Seamless Positioning

Positioning systems are often tailored towards a specific environment or use case. An indoor positioning system is designed to work indoors using a variety of algorithms, while an outdoor positioning system is used in the open air. With seamless positioning, we want to create a seamless transition from one positioning system to another [86]. This seamless transition ensures that users can have a continuous and uninterrupted positioning as they move between different indoor and outdoor environments, as illustrated in Figure 2.11.

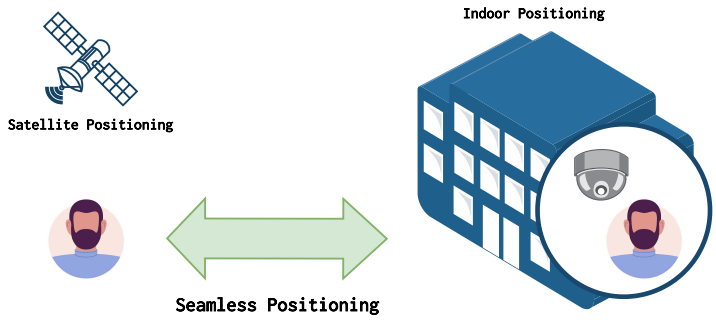


Figure 2.11: Illustration of seamless positioning between satellite positioning and indoor positioning

One of the main challenges to achieve this is the *handover* of the tracking from one positioning system to another. In a use case where a user walks from one building to another, the handover happens between the indoor positioning systems from the two buildings and optionally also an outdoor positioning system that tracks the user in between the two buildings [87].

The concept of creating seamless positioning systems is related to hybrid and interoperable systems due to the requirement to link from one to the other. To provide a seamless transition, both positioning systems must operate with interoperable data to correctly identify the same tracked object.

2.2.3 Indoor Landmarks

Indoor positioning systems can utilise a wide range of techniques within an indoor environment. One of the common methods to compute a position or location indoors is the use of visual, acoustic or electromagnetic *landmarks*. These landmarks prevent the need for a complicated setup and are less prone to changes in the environment as opposed to other techniques that utilise the existing infrastructure. However, they also require prior knowledge of the locations of these landmarks, which often results in the use of a database that is inaccessible to other systems.

Ever since the release of Bluetooth Low Energy (BLE), BLE beacons have been one of the more prominently used techniques for indoor positioning [88]. For their use in indoor positioning systems, they are often installed as small battery-powered devices in rooms, hallways or other key areas [89] but can also be embedded in smart devices [90]. During the installation, the beacon's physical location in the building is linked to its advertised identifier. Despite the advances in indoor positioning techniques that require no changes to the infrastructure, Bluetooth beacons are still among the most popular options for implementing indoor positioning systems due to their compatibility with various smartphone operating systems. Bluetooth beacons send out an RF-signal that can be used to determine the distance to each of these beacon landmarks.

One of the simplest positioning techniques using beacons is cell identification (see Section 2.3). With this technique, the proximity to a single beacon or a group of beacons is used to determine the receiver's position [54]. We assume that the beacons with the strongest signal strength are the closest. When multiple beacons are in range, multilateration can be used to compute the absolute position based on three or more beacons. In the context of positioning systems, the known fixed location of beacons is used to determine the location of a user scanning for these beacons.

In the Bluetooth Core Specification version 5.1, a new method for location tracking was introduced using the angle of arrival (AoA) or angle of departure (AoD) [91]. The specification added the ability for devices to generate data that could help receivers determine the angle of the signal. Advertisement data and the received signal strengths from the transmitted data can still be used to combine the angle with a distance approximation, offering better location tracking. Regardless of the used technique, Bluetooth-enabled devices are capable of computing their relative location between one or more beacons but require knowledge of the location of these beacons in order to compute their absolute location within a building.

An alternative to this setup is a positioning system that uses Bluetooth receivers with a fixed location. These receivers scan for moving *beacons* [92] while the transmitter is a moving beacon with an unknown location. Nevertheless, the same positioning techniques can be applied.

Modern phones have started to support Ultra-wideband (UWB). Other than BLE, UWB beacons provide more accurate positioning due to their higher precision in distance measurement. Ultra-wideband technology operates by transmitting short pulses at very high frequencies, allowing for accurate time-of-flight measurements. This enables UWB beacons to determine the distance between the transmitter and receiver with high precision, making them suitable for applications requiring centimetre-level accuracy, such as asset tracking or navigation in complex indoor environments.

2.3 Positioning Techniques and Algorithms

Different technologies and algorithms are used to calculate a position or combine multiple positioning methods. We provide an overview of some of these prominent techniques to later allow us to determine a generalised method to represent the input and output data of these algorithms, as defined in RQ1.1 and RQ1.2. A positioning technique consists of sensor data and an algorithm that processes this data using various techniques.

In this section, we focus on the most common positioning techniques and algorithms that are used in positioning systems, or more specifically, indoor positioning systems. While the techniques and algorithms discussed in this section are not exhaustive, they represent the foundational methods used in the field of indoor positioning and are primarily detailed to provide us with the different types of data that should be considered. Moreover, by investigating different techniques and algorithms, we can also consider the overlap of data and processing.

2.3.1 RF-based Positioning

Radio Frequency (RF)-based positioning techniques rely on the use of electromagnetic waves to estimate an object's position. This can include technologies such as Wi-Fi-based positioning, Bluetooth Low Energy (BLE) beacons, Ultra-wideband beacons or even cell towers. When these technologies transmit and receive information, the receiver can obtain additional information such as the Received Signal Strength (RSS) , representing the signal strength (indicated in *dBm*) received by the receiving antenna.

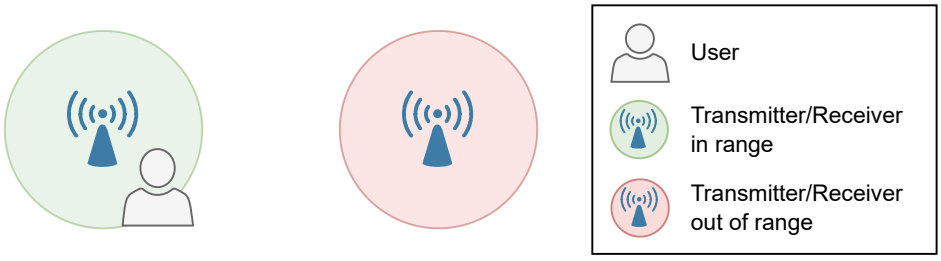


Figure 2.12: Radio Frequency Cell Identification

The most basic method of RF-based positioning is Cell Identification (CI) . This method involves dividing the area into cells as depicted in Figure 2.12, each with its unique identifier. The position of the object is then estimated based on the cell it is currently in. The estimation of the cell is based on the strength of the signal compared to the signal of other transmitters. Cell identification is often used in combination with other positioning techniques, as it can provide a general area before other techniques pinpoint a more precise position.

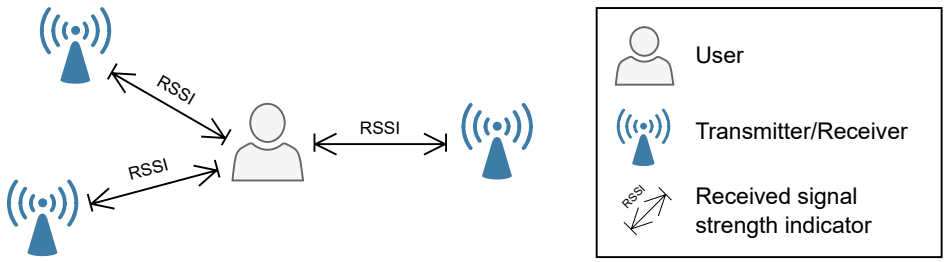


Figure 2.13: Radio Frequency Received Signal Strength Indicator

An extension to CI-based positioning is RSS-based positioning. Unlike traditional CI-based positioning where only the strongest signal strength is used, this technique estimates the position of an object based on the strength of the signal received from multiple transmitters, as illustrated in Figure 2.13. By comparing the signal strengths from different transmitters, the position of the object can be calculated using multilateration or trilateration. RSS-based positioning is more accurate than CI-based positioning but requires more complex algorithms to process the received signals and estimate the position accurately. The received signal strength is subjective to noise from the environment, multi-path effects, and signal interference, which can affect the accuracy of the position estimation.

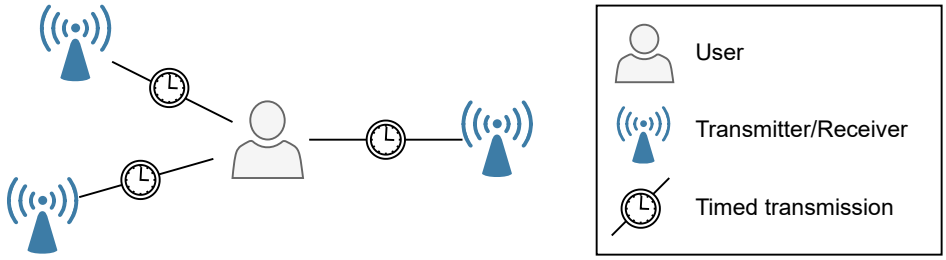


Figure 2.14: Radio Frequency Time of Arrival and Time Difference of Arrival

To solve the issues of signal strength noise, Time of Arrival (ToA) and Time Difference of Arrival (TDoA) techniques can be used. These techniques estimate the position of an object based on the time it takes for the signal to travel from the transmitter to the receiver, as shown in Figure 2.14. By measuring the time difference of arrival from multiple transmitters with a known position, an object's position can be calculated accurately. This type of RF-based positioning does not suffer from signal interference making it a more reliable method for indoor positioning applications. However, it requires precise time synchronisation between the transmitters and receiver to accurately calculate the time of arrival. Additionally, the accuracy of ToA and TDoA techniques can still be affected by factors such as signal reflections, obstacles in the environment and clock drift [93].

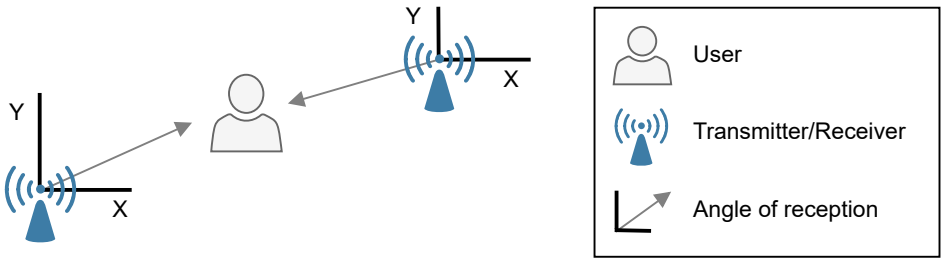


Figure 2.15: Radio Frequency Angle of Arrival

In TDoA, ToA and RSS-based techniques, the distance or estimated distance was used to determine a position. Alternatively, the angle of an incoming signal can be used to determine from what angle it was transmitted. Figure 2.15 depicts the concept of radio frequency Angle of Arrival (AoA) , showing how the angle of arrival of radio frequency signals can be used to determine the position of an object. This technique is commonly referred to as triangulation where the position of an object is based on the angle at which the signal arrives at different antennas. Unlike trilateration or multilateration, which uses the (approximated) distances between transmitters, triangulation uses the *angles* in which the signals arrive to determine the location. By combining data from multiple antennas, a more accurate position estimate can be obtained. A variant of AoA is Angle of Departure (AoD) where the transmitter can encode information to determine from which angle it was sent. While these techniques can obtain an accurate position, it does require multiple antennas for each transmitter or receiver to determine the angle.

2.3.2 Fingerprinting

Fingerprinting, as the name suggests, is the creation of a unique *fingerprint* at each location in the environment. Imagine being blindfolded in a building and hearing sounds from various machines. You hear the copy machine in the distance, the humming of an air conditioner overhead and a co-worker chatting. All of these sounds combined provide a unique fingerprint that can be associated with a location. In the *offline* stage of a fingerprinting algorithm, fingerprints are associated with a location and stored in a database. During the *online* stage, the database is used to determine to location based on sounds that are heard or other sensor data.

Machines can create these fingerprints as well. Like in the example above, they can be based on sounds or even ambient background noise [94], but they can also be created from data humans cannot detect, such as RF signals or magnetic interference. While with multilateration we only need information about the position of the used RF landmarks, fingerprinting requires a calibration for all possible positions in the tracking area [95]. A fingerprint of the sensor data at a given provided position is

created during the offline stage. Later, these stored fingerprints are used during the online stage to reverse the sensor data into a position.

Fingerprinting can be performed on a variety of sensor data types, including RSS, ToA, TDoA, and AoA data. Each type of sensor data provides unique information that can be used to create fingerprints. For example, RSS data can be used to create a fingerprint map based on signal strength measurements at different locations within the tracking area while ToA and TDoA data can be used to create fingerprints based on the time it takes for signals to travel between transmitters and receivers.

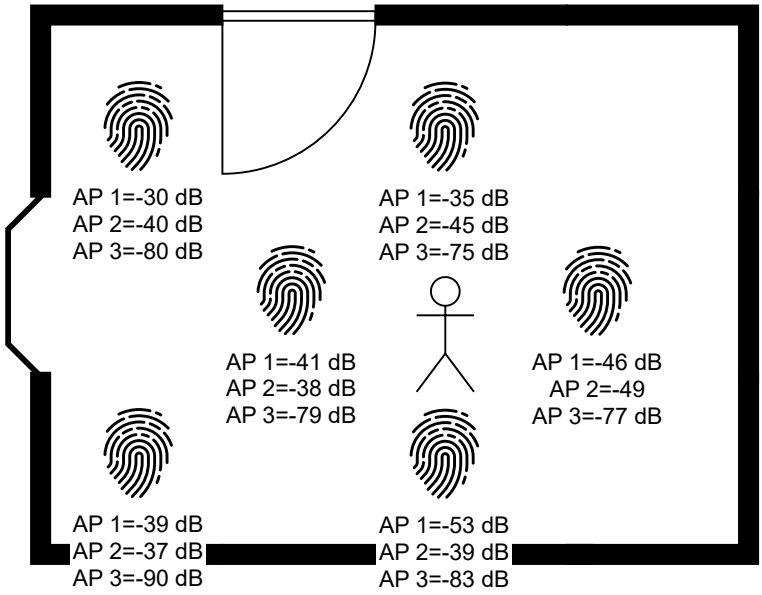


Figure 2.16: Fingerprinting of three access points with an unknown location

During the online stage, the sensor data collected in real time is compared to the stored fingerprints to determine the most likely position of the object. The matching process can be based on various algorithms such as nearest neighbour, k-nearest neighbour [96], machine learning algorithms like support vector machines or neural networks [97]. Despite the computational complexity involved in fingerprinting-based positioning, it offers high accuracy and robustness in indoor environments where traditional positioning techniques may not perform well due to signal variability.

Figure 2.16 illustrates the fingerprinting of an indoor environment with a limited amount of *features*. Features are the most noticeable characteristics of the sensor data that are used to create the fingerprints. In the example, we have three access points with an unknown location and multiple fingerprints that recorded the signal strengths of these three access points. In the online stage of the positioning system, the signal strengths are compared to the fingerprints. Based on the fingerprinting

algorithm, a location is chosen from the fingerprint that most closely matches the signal strengths or multiple fingerprints are combined to interpolate a location in between the recorded fingerprints.

2.3.3 Magnetic Positioning

Magnetic positioning, also called geomagnetic positioning, is a positioning technique that uses the disruptions of the magnetic field within a building to fingerprint locations [75, 76]. The earth has a magnetic field that can be influenced by various factors such as building materials, electrical equipment, and structural components. These factors can create unique magnetic signatures at different locations within a building, which can be captured and used for positioning.

In the offline stage of magnetic positioning, magnetic field measurements are taken at various locations within a building and stored as fingerprints in a database. During the online stage, real-time magnetic field measurements are compared to the stored fingerprints to determine the current location of an object.

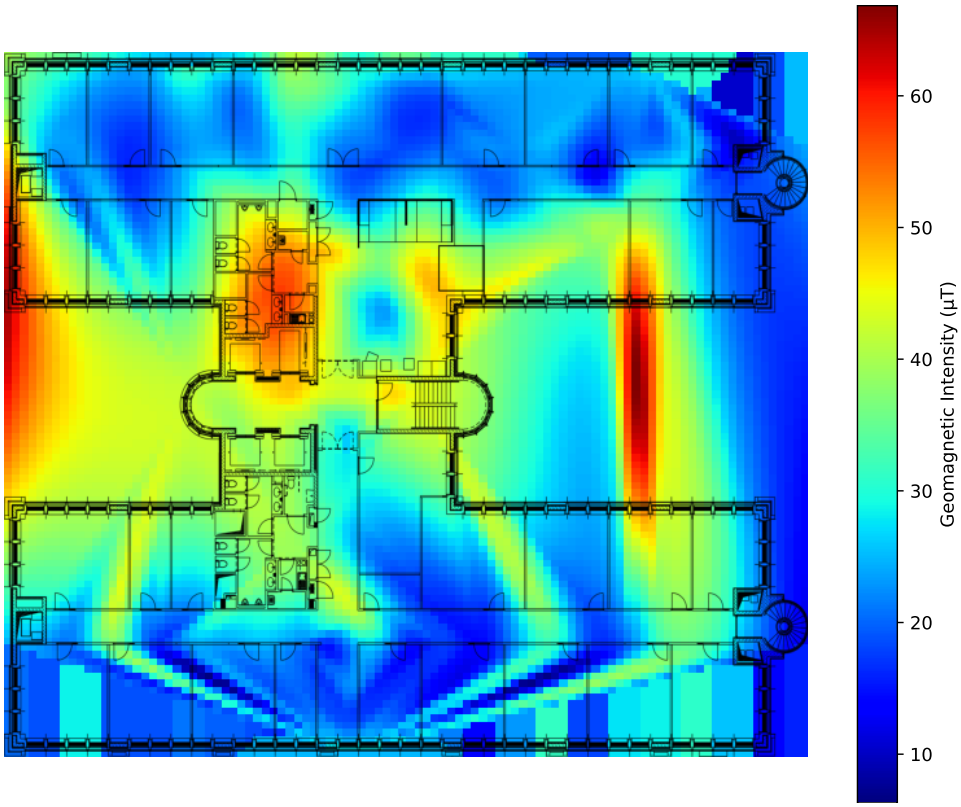


Figure 2.17: Geomagnetic intensity of our office building with cubic interpolation

Figure 2.17 shows the geomagnetic fingerprint of our office building with a dataset we recorded in 2021 [35]. This geomagnetic intensity is visualised as $\sqrt{magX^2 + magY^2 + magZ^2}$ where $magX$, $magy$, $magZ$ represent the magnetic intensity measured on three axes in μT. Similar to fingerprinting, a set of features (i.e, a small trajectory) can be matched against the geomagnetic fingerprint of a building or floor.

Sensor data from a magnetometer is susceptible to *hard iron* and *soft iron* effects. Hard iron effects interfere with magnetic fields that cause a constant offset in the readings, such as other metallic objects near the magnetometer. Soft iron effects distort the magnetic field in a non-uniform manner. These effects need to be corrected for accurate magnetic positioning through calibration of the sensor [98]. This type of technique demonstrates the need for user actions in positioning systems. When designing a conceptual framework for processing sensor data, we should consider the need for calibration and user actions.

2.3.4 Noise Filtering

Positioning techniques often have to process inaccurate sensor data to obtain a position. Techniques such as RF-based positioning, as shown in Figure 2.18, use signal strengths that can fluctuate depending on the distance. To solve these inaccuracies, sensor data should be filtered, which can be done through different noise filters. Similar to dead reckoning, a noise filter often requires knowledge of previous sensor readings and positions to predict the next result.

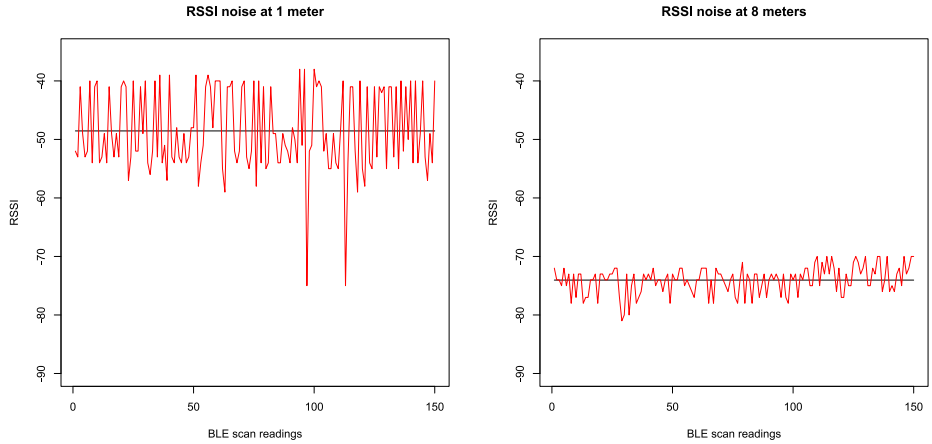


Figure 2.18: RSSI noise (in dBm) at 1 metre and 8 metres [99]

Noise filtering is one of the main requirements of our hybrid positioning system. The reason why we want to combine multiple technologies or algorithms is to reduce errors and noise filtering is the key component in realising this error reduc-

tion of positioning data. Note that individual positioning methods such as object recognition or dead reckoning may want to perform different types of noise filtering algorithms tailored to their data.

We also consider noise filtering algorithms that consist of a calibration phase, such as the calibration of a magnetometer as detailed in Section 2.3.3. These algorithms require user interaction and the storage of *calibration data* that must be used at a later stage.

2.3.5 Machine Learning

This type of algorithm includes several machine learning algorithms that can be of aid during calibration as well as positioning. These algorithms require training during the offline stage with the results being deployed during the online stage.

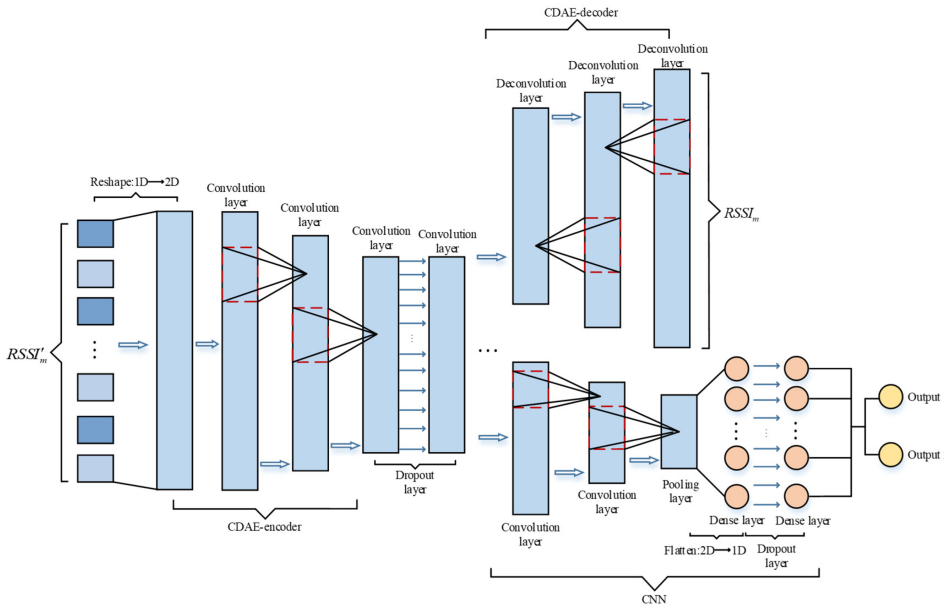


Figure 2.19: CCpos [97] Wi-Fi RSSI fingerprinting neural network model

Figure 2.19 shows the neural network model of CCpos [97]. Other than regular fingerprinting techniques introduced in Section 2.3.2, the model trains itself to find patterns and features within the RSSI fingerprints. Training is done using an autoencoder. Such a model trains the input data of the encoder to be equal to the decoder output. The dimensions of the middle part of the autoencoder become smaller to force the encoder part to extract the useful features that can then be decoded by the decoder. Once the main model is trained, the decoder is removed, and the model is trained using the encoder to match RSSI readings to a position.

2.3.6 Computer Vision

Visual positioning techniques use image sensors to determine the position of the object the sensor is attached to (e.g., Visual SLAM [50]) or the position of objects in its field of view. Such visual positioning methods have to be able to detect and track objects between multiple frames, camera angles or positions. The visual SLAM technique, as shown in Figure 2.20, creates a 3D model of the environment using images captured by the camera. By analysing the visual data and optional depth information, the system can determine the spatial relation between visual features while mapping the complete environment.

Figure 2.20: Visual SLAM 3D model of a part of the hallway in our office

Alternatively, visual positioning techniques can also identify and track the position of objects. With Multi-Target Multi-Camera Tracking (MTMCT) [49] the tracked object is moving within the field of view of one or more image sensors.

2.3.7 Dead Reckoning

Dead reckoning calculates the current position based on the previous known position and the velocity or acceleration that is applied to that position [72, 100]. While the accuracy of dead reckoning is not ideal, it can be used to improve other positioning techniques such as GPS.

Variations on dead reckoning exist that compute the raw movement information to a higher level of abstraction, such as Pedestrian Dead Reckoning (PDR) which specifically focuses on tracking a pedestrian's movements. PDR uses sensors like accelerometers and gyroscopes to estimate step counts and direction changes, providing a more detailed positioning approach compared to regular dead reckon-

ing. Figure 2.21 illustrates a basic example of dead reckoning starting from an initial point at timestamp 0. Based on the speed and orientation, the next data points can be calculated.

Dead reckoning is not limited to accelerometer data. Variations exist such as visual- or geomagnetic dead reckoning. Visual dead reckoning detects movement from one video frame to the other [101] while geomagnetic dead reckoning uses changes in the magnetic field to determine movement.

2.3.8 Sensor Fusion

Positioning algorithms can range from simple multilateration to fingerprinting or even artificial intelligence. In hybrid positioning systems, sensor fusion combines data from multiple data streams to improve the accuracy and latency of a positioning system. By integrating data from different data streams such as GPS, IMU, LiDAR, and cameras, the system can compensate for the limitations and errors of individual sensors.

Sensor fusion can occur at a low or high level [102]. Raw sensor data, such as IMU sensors or the relative signal strength from a transmitter, can be fused in noise filtering algorithms. On a high level, calculated or provided positions (i.e., by third-party positioning systems) with a certain predicted accuracy can be combined using linear regression, heuristic weighted averages or any *decision fusion* algorithm.

Figure 2.22 illustrates the use of a weighted sensor fusion algorithm to combine Wi-Fi fingerprinting, RSSI multilateration and cell identification using Bluetooth Low Energy signals and finally also dead reckoning. Dead reckoning on itself is an imprecise technique but offers a continuous estimation of a moving object's position. By combining this dead reckoning with other sensors, we can create a more accurate and reliable trajectory.

Low-level Sensor Fusion

Low-level sensor fusion involves the integration of raw sensor data to filter out noise and improve the quality of the measurements. This type of fusion is commonly used

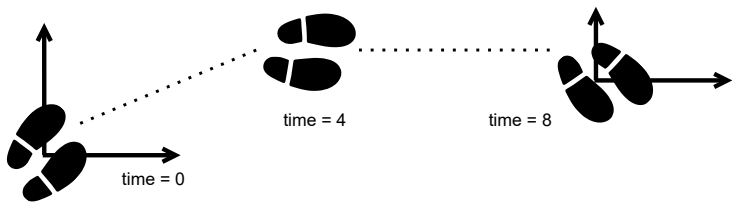


Figure 2.21: Dead reckoning demonstration

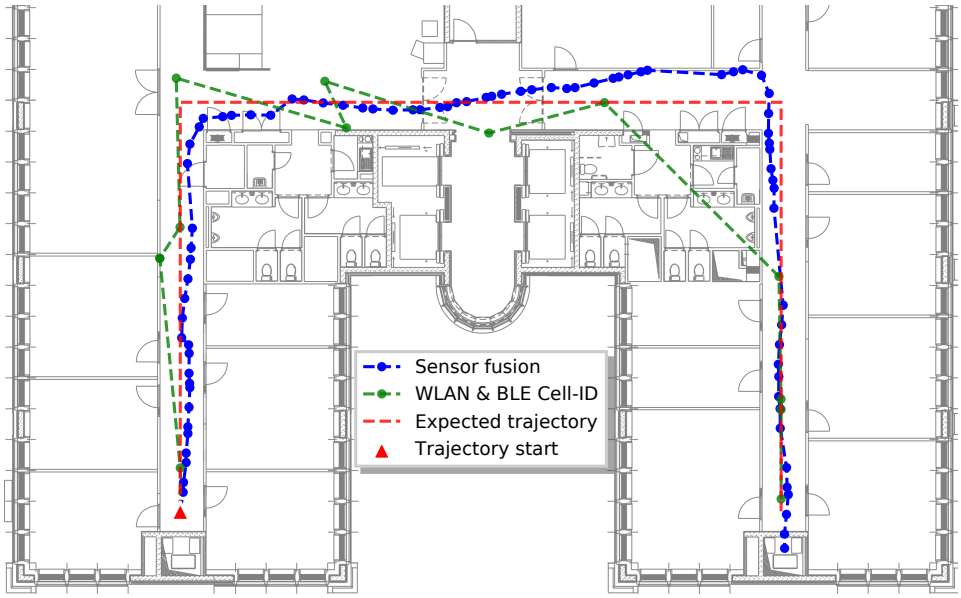


Figure 2.22: Sensor fusion of a trajectory (blue) based on WLAN, BLE and IMU data

in inertial measurement units (IMUs) where data from accelerometers, gyroscopes, and magnetometers are combined to accurately track the position and orientation of an object in real-time.

Noise filtering algorithms such as Kalman filters or complementary filters [103] are often used to fuse data from multiple sensors and compensate for any drift or errors that may arise during measurement.

High-level Sensor Fusion

Unlike low-level sensor fusion, high-level sensor fusion involves the fusion of two or more computed results from individual algorithms or sensors. Decision fusion algorithms are a type of high-level sensor fusion, where the goal is to combine multiple sources of information by deciding or *weighing* the importance of each information source.

Recent research on indoor and outdoor positioning systems makes use of artificial intelligence to process sensor data [97, 104, 105]. These types of systems often implement a complex neural network that uses training data to learn how to process data in the online phase of a positioning system. While deep Kalman filters [106] can be used to perform sensor fusion, it is often difficult to perform sensor fusion of unknown data that is already processed by AI-based data processors. One solution for this issue is to treat these processors as processors that provide a high-level output with a certain precision and accuracy.

2.4 Interoperability of Data and Services

In this dissertation, we aim to design *interoperable* positioning systems. Interoperability is defined as the ability to access, read and understand data from multiple data sources [107, 17, 18]. When software developers create applications that should be interoperable, the data that is processed by these applications must be made available to other third-party services (i.e., accessibility). Next, third-party applications should be able to read this data in a common file format or standardisation. Finally, after being able to access and read the data, third-party applications should be able to understand the meaning of the data in a way that is indifferent to what the producer of this data intended. Accessibility and readability are often referred to as syntactic interoperability while the understanding of the data is often explained as semantic interoperability [30].

In addition to syntactic and semantic interoperability, *processing interoperability* [108] describes how semantic data is processed in a system. It is also referred to as pragmatic [109] or behavioural interoperability [110] and provides a system with more context on how data was, or will be processed. With our aim to design interoperable positioning systems, we will focus on achieving all three aspects of interoperability: syntactic, semantic, and processing.

Interoperability of positioning systems is not a new concept. Global Navigation Satellite Systems (GNSS) such as Galileo are designed to be interoperable with the widely used GPS standard via RF signals. Likewise, interoperability of location-based services (LBSs) has been considered since the early days of ubiquitous LBSs [111]. Specifications and standards included in OpenLS by the Open Geospatial Consortium (OGC) have contributed to the development of LBSs. OpenLS is an open location service architecture that defines how OpenLS-compatible applications should interface with a server to retrieve information [112]. Despite the standards using common data structures, there is still a lot of freedom that breaks interoperability between systems. In addition, an LBS only aims to retrieve basic location data; a positioning system can contain much more information that is not included within the OpenLS standards.

Figure 2.23 illustrates the architecture of an OpenLS implementation. The GeoMobility Server (GMS) illustrated in blue at the bottom implements the OpenLS core services, which include but are not limited to geocoding services and routing. The GMS can communicate with Gateway Mobile Location Centre (GMLC) and Mobile Positioning Center (MPC) services through a *Location Interoperability Forum (LIF) API*. This API is built on top of the Mobile Location Protocol (MLP) and provides a common protocol for positioning and location retrieval. The service platform depicted in the centre handles the authentication, provisioning and context management when communicating with the GMS. The architecture does not require OpenLS and GMS servers to be decentralised. In addition, requests

still rely on interfaces that are not fully defined in the standard. Efforts have been made to design a GMS server using the Semantic Web [113]. However, in these scenarios, one often has multiple of these services for tracking only one particular entity.

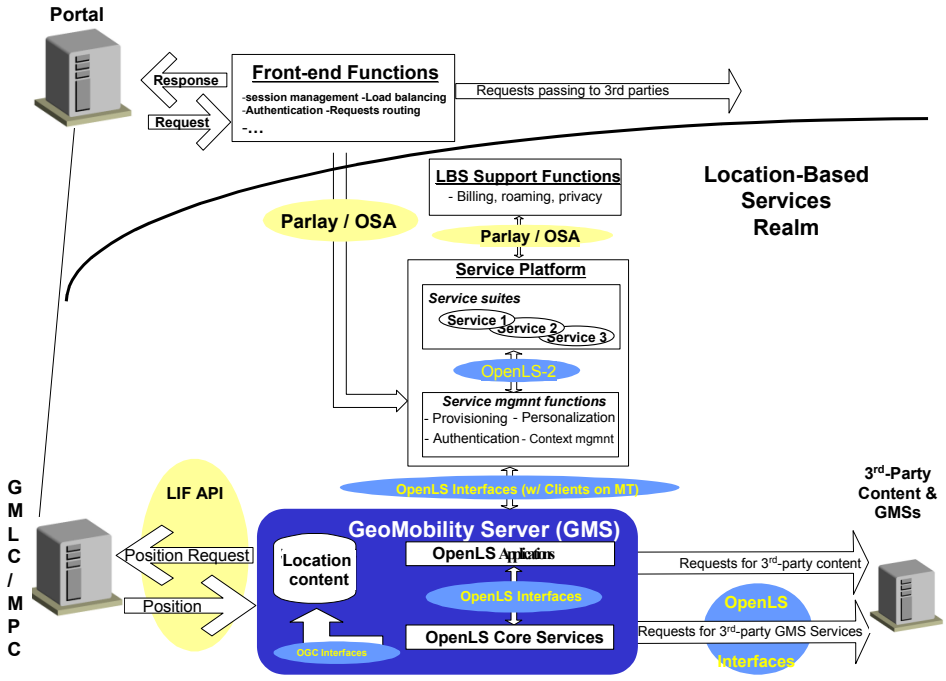


Figure 2.23: OpenLS architecture [112]

The work of Furfari et al. [26] proposes a design architecture for enabling the discovery and interoperability of Location-based Services (LBSs). In this design, they propose a standardised agreement of the discovery of such systems and their capabilities, the workflow description and agreement on the user interface. Their design goal closely relates to our aim towards interoperable and discoverable indoor positioning systems. In earlier work, Furfari et al. [25] provide a taxonomy for standardising indoor positioning systems. In this taxonomy, they define how interoperability of location-based services should be defined in order to promote the integration and collaboration of indoor location-based services. However, they provide no implementation or solution to this interoperability. In their prototype that provides discovery and interoperability [26], they provide a protocol that uses Open Database Connectivity (ODBC) drivers to interface with different systems.

While the solution does not solve the fragmentation of user data across different services, it does solve the interfacing problem of accessing data from these systems. However, interfacing with the positioning service relies on standardised functions that map these functions to ODBC drivers. These functions are never defined

and are based on common use cases and ISO 18305:2016 [22], which does not standardise the data exchange.

Creating interoperable applications requires the definition of context and standards for data exchange, such as using common data formats like JSON or XML. Additionally, implementing standardised interfaces and APIs can facilitate the seamless integration of different services and data sources. By promoting interoperability, developers can ensure that their applications can easily communicate and share information with other systems, enhancing the overall functionality and efficiency of services such as location-based services. Furthermore, interoperability can lead to the development of more robust and modular applications that can adapt to a wide range of scenarios and user requirements.

2.4.1 Location Data

The interoperability of location data is critical for the seamless functioning of independently running hybrid positioning systems. Different positioning technologies often use different formats and protocols to represent data, including location information. In order to effectively combine these technologies into a single positioning solution, it is important to have a standardised format for representing and exchanging this data.

Efforts to ensure *interoperability* of location data are not new. Standardisations such as the World Geodetic System (WGS) began to be used in the late 1950s to ensure a common reference system for location data across different applications and services using the WGS-84 standard [65]. Other standardisations such as Earth-centered Earth-fixed (ECEF) [114] or the North American Datum (NAD) also offer a fixed frame of reference that can be used to express the location data. On a low level, this generally means that location data can simply be expressed as two-dimensional or three-dimensional coordinates in a well-described frame of reference. Without a frame of reference, the coordinates are simply meaningless numbers.

Standardisation and reference frames ensure semantic interoperability, but they do not solve syntactic interoperability, since coordinates and the used reference frames can be expressed in different ways. One approach to achieve syntactic interoperability in location data is through the use of standardised data formats such as GeoJSON [115]. This format provides a common structure for representing geographical data in JSON format, allowing different positioning technologies to easily exchange and interpret location information. Figure 2.24 shows several different geometries that can be expressed in GeoJSON, such as points, lines, and polygons.

Well-known Text Representation (WKT) is a markup language to write a vector geometry such as a point, line or polygon [116]. The format was defined by the

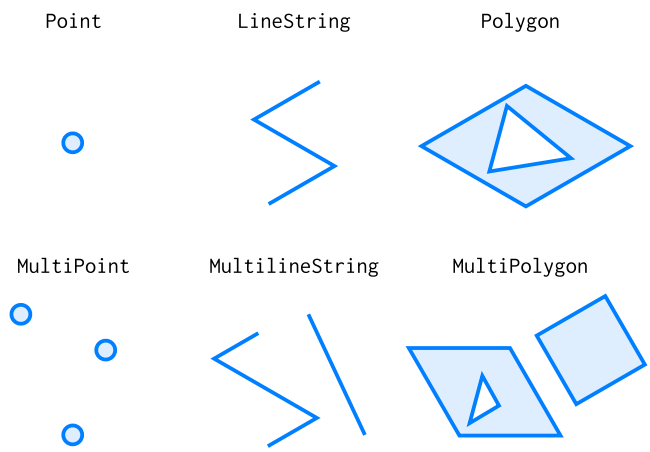


Figure 2.24: Different vector geometries available in GeoJSON

Open Geospatial Consortium (OGC) and has since become a widely used standard for representing geometries in a human-readable way. WKT only provides a markup language for geometries but does not allow specifying the reference system that is used. The format is extended with variations such as the Extended Well-known Text Representation (EWKT) which adds a textual Spatial Reference System Identifier (SRSI) before the existing WKT format [70].

Similar to WKT is the Geography Markup Language (GML) format which is also created by the OGC. GML is a richer format that allows for the representation of complex geographical features and their relationships using the XML syntax. By default, it offers XML attributes to specify the spatial reference system [117].

2.4.2 Indoor Environments

Indoor positioning systems often require knowledge of the environments in which they are deployed. In the case of positioning techniques such as trilateration, triangulation or cell identification this information is the location of access points and beacons. However, with more complex positioning techniques or navigation systems, a more detailed description of the indoor layout is required.

The environment data that needs to be discovered for a positioning system can be referred to as a digital twin [118]. A digital twin can provide additional context about its physical counterpart, enabling systems to reason more about the available data and interactions possible with a physical object. The World Wide Web Consortium (W3C) describes the concept of the Web of Things (WoT) , combining virtual

and physical *Things* connected through the Web. A digital twin of an environment can be modelled as a virtual 3D environment as shown in Figure 2.25 or a semantic description of the geometry of the environment [119]. In the context of designing interoperable positioning systems, a digital twin of the environment can be used to more accurately interpret sensor values (e.g., by analysing obstructions) or to provide more context to the user on their position in navigation applications.

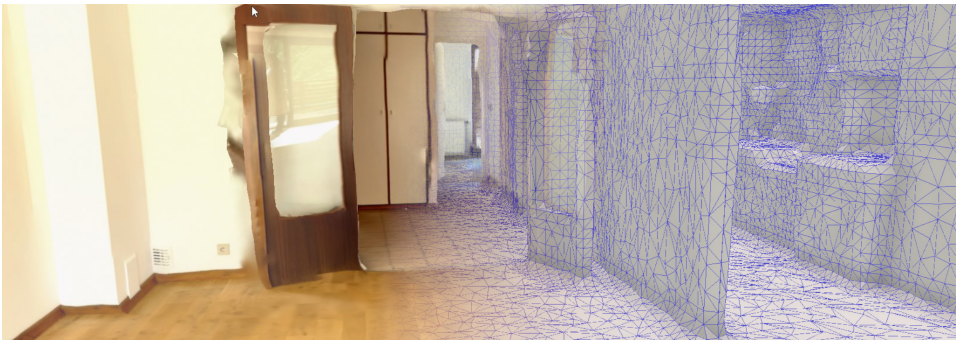


Figure 2.25: Example of the digital twin of an environment

Many indoor map formats and specifications exist that each tailor to a specific use case. Indoor Mapping Data Format (IMDF) [120] is a mapping format developed by Apple Inc. based on GeoJSON. It adds additional data to the GeoJSON features to specify the type of feature and associated data. IMDF is supported by Apple Maps and Google Maps. The format offers a basic representation of indoor spaces that can be expanded with additional details such as doors or relationships between features. In addition to IMDF, there are other open data specifications such as OGC's IndoorGML [121], which specifically focus on the layered representation of indoor environments, including the topological layer as well as the Wi-Fi coverage in an indoor environment.

OpenStreetMap uses its own tagging schemes called the Simple Indoor Tagging (SIT) scheme and Simple 3D Buildings (S3DB). These tagging schemes offer a simplistic representation of an indoor environment, and are, as the name suggests, meant to simply define an indoor environment rather than provide detailed information about it. The Simple Indoor Tagging (SIT) scheme focuses on basic indoor features such as rooms, corridors, entrances, and exits, while the Simple 3D Buildings (S3DB) scheme provides information about the outside 3D layout. While the schemes are open, they are not extensible, making it impossible to provide more context beyond the *simple tags*.

While these standardisations offer a data format for representing indoor environments, the data of these environments is often service-centric, making it difficult to integrate with other systems or applications. In this dissertation, we primarily focus on the essential indoor geospatial data necessary to facilitate the interoperability of

indoor *positioning* systems. However, future work can also benefit from enhancing the interoperability of indoor environments to facilitate indoor *navigation*, which also requires output to users.

2.4.3 Semantic Web

One of the common solutions to enable the accessibility and readability of data in interoperable applications is to make this information available on the Web. Information can be published using various interfaces such as RESTful APIs [122], SOAP [123] or linked data [124]. While RESTful APIs and SOAP typically expose data through predefined endpoints and formats tailored to specific applications, Linked Data follows Semantic Web principles, using standardised Web technologies (e.g., URIs, RDF, and HTTP) to interlink and describe data in a machine-readable way, thereby enabling automatic data integration and discovery across diverse sources.

Linked data creates a *web of information* over the World Wide Web (WWW) that allows everyone to define new concepts by referencing other concepts found on the Web. A machine that understands linked data can traverse and make connections between different sources of information, leading to a more interconnected and comprehensive understanding of the data.

One of the key strengths of linked data is its ability to link multiple standardisations and data formats together. With positioning systems containing various types of data, ranging from environments, location data, and the algorithms surrounding it — interoperable positioning systems benefit from using linked data to describe this knowledge. Standardisations such as WKT, GML, and GeoJSON have found their way to linked data representations and are commonly used when expressing location data. While many efforts have been made to enable the interoperability of geographical location data, a framework for ensuring both syntactic and semantic interoperability across different positioning systems still needs to be developed.

The Semantic Web and linked data are concepts that are formally introduced by the founder of the web, Tim Berners-Lee [124] in the late 1990s. Linked data is the collection of data and vocabularies connected over the World Wide Web (WWW), where new concepts are described by utilising existing vocabularies located on the Web. This creates an interconnected *web of linked data* that can be accessed, queried and expanded.

Information on the Semantic Web can be expanded by adding new vocabularies or knowledge graphs to the Web. This interconnected network of data allows for more comprehensive and meaningful information retrieval, enabling computers to not only access data but also understand its meaning, relationships and additional context.

In our goal to design interoperable positioning systems, linked data is used for its ability to generalise concepts while making these concepts and data publicly available via the Web. This provides both semantic interoperability by leveraging the use of vocabularies, and syntactic interoperability by enabling the exchange of data through the HTTP protocol.

One of the potential drawbacks of semantic data is the overhead involved in describing information. When a system has a prior understanding of the expected data, it can leverage this knowledge to compress the information into the bare essentials — this is essentially how compression algorithms work. However, when a system does not know the data, the data itself should contain information about its purpose. In our OpenHPS framework, introduced later in Chapter 3, we developed a communication layer that uses protocol buffers to convert data to binary information. Without knowledge about the format of this data, it is not useful for any other system. With semantic data, we want to prevent the need for prior knowledge by handling self-describing data. While this ensures that each system understands the information, it does create additional overhead of information that has to be stored.

Linked data is structured interlinked data. It is often represented as Resource Description Framework (RDF) graph data, which is comprised of triples consisting of a subject, predicate and object. A predicate creates a directional relationship between a subject and an object. The object can be a subject itself when it has predicates linking to other objects.

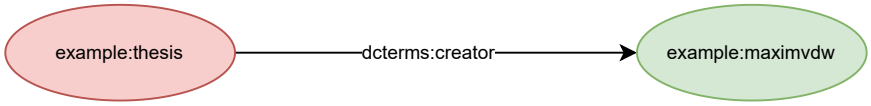


Figure 2.26: Subject and object connected via a predicate

Figure 2.26 illustrates an example that shows a subject (indicated in red) for representing this thesis document. The object (indicated in green) describes the author of the thesis. Both the subject and object are connected using the directional predicate `dcterms:creator` which describes the object as the creator of the subject using the public Dublin Core vocabulary [125].

Named concepts or terminologies such as the subject, object and predicate in the example of Figure 2.26 are *named* (i.e., identified) using a uniform resource identifier (URI). When this named concept is accessible via the Web using its URI, we can follow this URI to obtain more information. In our example, these URIs are denoted using a prefix (i.e., `example:` and `dcterms:`).

Objects can also consist of literals, in which case they cannot be used as a subject in another triple. In Figure 2.27, we expanded on our previous example with an atomic value indicating the title of the thesis and the name of `example:maximvdw`.

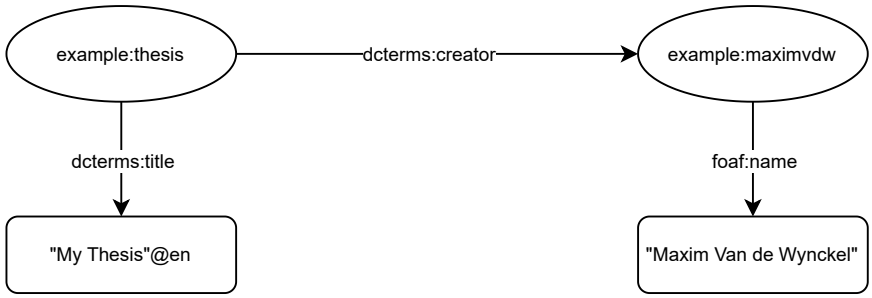


Figure 2.27: Subject and object connected via a predicate

Together, sets of triples create a graph of linked data consisting of nodes and directional relations between these nodes. Atomic values are either strings with a language tag (i.e., a tag that indicates in what language the value is) or strings with a data type that indicates the type of literal.

Linked data documents on the Web contain a collection of triples within the same *base* URI. A base URI directs to the document, while a resource URI directs to a specific subject within a document. Preferably, these documents and resources can be fetched over the Web by accessing their URI.

```
example:  <https://phd.maximvdw.be/mythesis.ttl#>
```

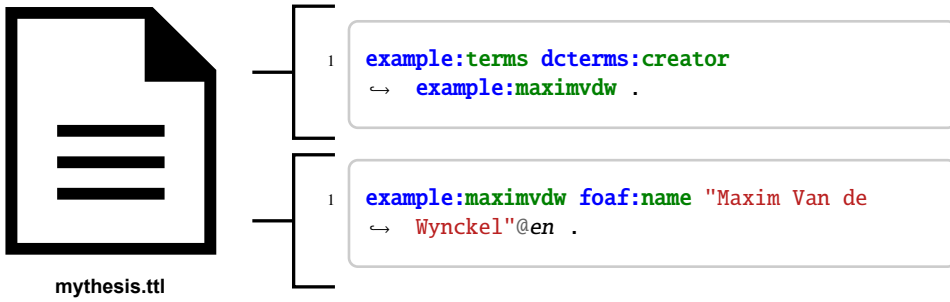


Figure 2.28: Linked data document containing triples

Figure 2.28 illustrates a linked data document with multiple triples. This linked data document contains triples representing different subjects, predicates, and objects within the same base URI. Each triple forms a relationship between the subject and object, creating a network of interconnected information. By following the URIs of these resources, additional data can be retrieved and further expand the knowledge graph.

Querying of Linked Data

Similar to relational databases, linked data can also be queried to retrieve information [126]. The difference between querying a single relational database compared to querying a graph database is the ability to query other data sources the graph connects to. When querying linked data, each semantic concept can have its own URI that optionally can be accessed via the Web. When this is the case, querying can use link traversal [127] that queries over multiple resources spread over the Web, allowing systems to theoretically query the entire semantic part of the World Wide Web.

```
SELECT ?subject WHERE {
    ?subject dcterms:creator ?person .
    ?person foaf:name "Maxim Van de Wynckel" .
}
```

Listing 2.1: Example SPARQL query that returns the URIs of all subjects created by the person named “Maxim Van de Wynckel”.

Listing 2.1 demonstrates an example of a query executed over linked data that will return all the resource URIs of documents created by a person named “Maxim Van de Wynckel” using the `dcterms:creator` predicate. Such a query is called a SPARQL Protocol and Query Language (SPARQL) query and is executed on a graph of triples. While link traversal allows for querying over distributed semantic data, it also introduces challenges related to data consistency, availability and scalability. As the size and complexity of linked data graphs increase, the efficiency and effectiveness of query processing become more crucial [127].

SPARQL queries can include functions that are executed over the atomic values in the graph. These functions can be used to perform operations such as string manipulation or mathematical calculations. By leveraging these functions, SPARQL queries can be customised to retrieve specific information and perform advanced data analytics on linked data graphs.

```
SELECT ?feature ?geom ?dist WHERE {
  ?feature ex:hasGeometry ?geometry .
  BIND(geof:distance(?geom, ex:Maxim, uom:metre) AS ?dist) .
  FILTER(?dist < 500)
}
```

Listing 2.2: Example SPARQL query that finds all geometry features within 500 metres of Maxim.

The query engine that processes SPARQL queries can be extended to support additional functions. In the case of positioning systems, GeoSPARQL [128] might, for instance, extend the SPARQL engine with additional functions that perform spatial operations.

Listing 2.2 showcases a GeoSPARQL extension for calculating the distance between two geometries and filters the result to only list all geometries within 500 metres. The function `geof:distance` is a custom function added by the GeoSPARQL extension that calculates the distance within the SPARQL engine and is not present in SPARQL by default. By executing this query on a collection of geometric features spread over various knowledge bases, the SPARQL engine can retrieve all appropriate features.

Data Shapes

Having a vocabulary to describe data is one of the requirements to ensure data interoperability between systems. However, another important requirement is that the data is structured in a way that makes sense for the used vocabularies and applications. The Shapes Constraint Language (SHACL) and Shape Expressions (ShEx) [129] are two specification languages that represent data shapes. A data shape expresses the constraints that data graphs must conform to. These constraints define the structure, type, and cardinality of the information so data represented with vocabularies follows certain conventions. For example, when working with geographical coordinates, the latitude and longitude can be expressed as numbers. However, this number can only range between -90 and 90 for latitude and -180 to 180 for longitude. By defining a data shape using SHACL or ShEx, data can be validated to ensure it complies with these constraints before being used in applications or shared with other systems. These data shapes can aid developers in following certain conventions when representing their data, as well as providing other applications with a *blueprint* of how a certain application stores the data it produces. Similar to the data itself, these data shapes are described with RDF as well.

Vocabularies

An interoperable positioning system needs to offer interoperability for the input and output data, as well as a description of how this data is processed. In Chapter 2 we detailed how a hybrid positioning system uses high- and low-level sensor fusion to combine information data from multiple technologies and sources. To enable *process interoperability* on the data of a positioning system, these interoperable applications need to reason on the quality and origin of the data. Process interoperability entails that the processes used to manipulate data are interoperable in a way that describes them in detail, so another system could take over the process.

The Semantic Sensor Network (SSN) ontology, together with the Sensor, Observation, Sample and Actuator (SOSA) ontology [130, 131], are two ontologies that allow us to describe individual *observations* and the systems and procedures that were involved in obtaining these observations. Both ontologies and the various available extensions allow us to describe complex sensor networks.

```
:temp a sosa:ObservableProperty ;
    sosa:isObservedBy :temp-sensor .

:temp-1 a sosa:Obseration ;
    sosa:hasResult [ a qudt:QuantityValue ;
          qudt:numericValue 30 ; qudt:unit qudt-unit:DegC ];
    sosa:observedProperty :temp ;
    sosa:resultTime "2025-03-10T06:45:00.000Z"^^xsd:dateTime .
```

Listing 2.3: Observations of the SOSA ontology

Listing 2.3 demonstrates an example observable property of a temperature sensor with observation(s) for this property with a result. Each observation can have contextual information, such as the time it was observed and other information.

AI-based positioning systems such as CCpos [97] or CNNLoc [104] are often considered black-box implementations due to the sometimes unpredictable and unexplainable way in which output data is computed through deep neural networks. However, the mere fact that these can be explained also entails that they can be semantically described as individual data processors in an interoperable system. Semantic ontologies such as the Neural Network Ontology [132] can help to describe the models that a positioning system uses to compute the information, while ontologies such as our positioning system ontology [40] can describe common artificial intelligence models used for positioning systems.

2.5 Privacy and Transparency of Positioning Systems

When working with location data, the privacy of user data is one of the most important aspects to consider. Smartphone operating systems such as iOS and Android already offer a lot of transparency when applications request a location or sensor data that can be used to infer a location [133]. However, when working with proprietary positioning systems that either perform inside-out or outside-in tracking, users are often not aware of which data is retrieved and how this is processed.

Privacy covers various privacy concerns. Furfari et al. [26] discuss five main categories of privacy concerns related to location-based services. These include the *notice* and *consent* of users. Before a system commences the tracking, collection or sharing of location data, the user should be made aware and agree to this collection or sharing of information. The *purpose* of the data collection should be made clear to the user, ensuring transparency and trust in the system that is *accountable* for the data. Finally, users should have the ability to *access* and control their own location data, including the possibility of preventing a positioning system from accessing this data in the future.

Ensuring the privacy of location data when provided to third-party applications or services has multiple variations, each with its advantages and disadvantages. In most cases, the privacy of the location data is provided for location-based services that provide a high-level computed version of the location.

Location Privacy Protection Mechanisms (LPPMs) are a set of algorithms or mechanisms that attempt to protect the privacy of location data. They offer a set of lossless and lossy techniques to manipulate the data. Jiang et al. [134] identify three main categories of LPPMs that can be found in most public location-based services:

- **Obfuscation-based**: These types of mechanisms ensure privacy by manipulating the accuracy, precision and correctness of data outputted by a positioning system or a location-based service. This type of mechanism alters the raw location data in such a way that the actual location of the user is still preserved, but with added noise or distortion to protect the user's privacy. A simple example of an obfuscation-based LPPM in indoor positioning systems is the edge processing of a position to determine the general area (e.g., room) within an indoor environment, without exposing the accurate position within this area. This allows an application to determine that a user was in a certain office for a certain time, but not the movement of the user within this office.

- **Cryptography-based**: While obfuscation manipulates the data to ensure privacy, this is often not feasible depending on the use case and almost always causes a loss of information. Cryptography algorithms ensure that no data is lost but can only be accessed by authorised parties. This can be done by completely encrypting the data so that only trusted and authorised parties can access it, or by manipulating the location data in a way that it can still be processed without revealing the exact location.

- **Transparency-based**: Unlike obfuscation or cryptography, this type of mechanism does not rely on algorithms but instead relies on establishing a clear privacy policy and consent with the user who is being tracked. It can also include the possibility for users to see the data that is being collected.

Jiang et al. [134] also consider a fourth category, which they call *Cooperation and Caching-based mechanism*, where the communication of location-based data is reduced. Similar to obfuscation-based mechanisms, this also includes a loss of information.

While obfuscation, cryptography and transparency are considered feasible techniques to ensure privacy, they often come at the cost of the usefulness of the data. In the scope of our solution towards interoperable positioning systems, we did not implement any specific LPPMs in our positioning systems. However, we did consider the possibility for data to be manipulated by other services to comply with these LPPMs.

In 2025, we surveyed how users perceive the privacy and transparency of location data. Our aim with this survey was to determine the importance of location data amid various recent cases where location data was found to be sold or leaked. Based on the results, available in Appendix E, we discovered that users prefer not to share location data unless required. However, the majority of users do not read the privacy policies, which are in place to provide transparency on the type of tracking.

2.5.1 User Transparency

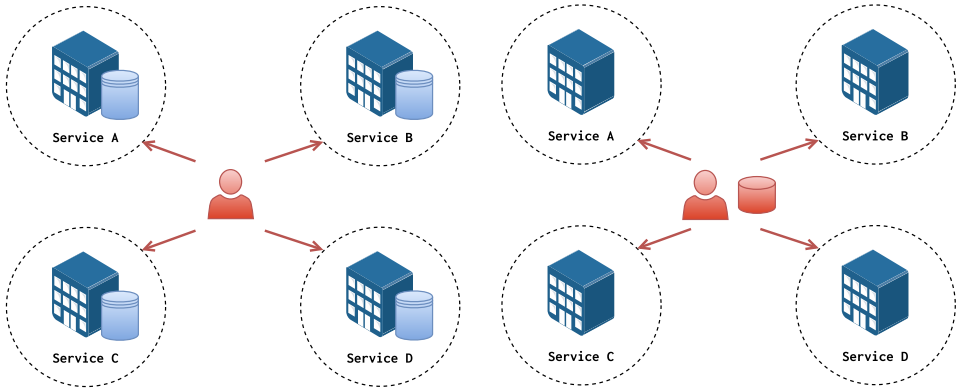
Privacy and security are important to consider when working with sensitive data, as is the case with location data. Current positioning systems often have a black box implementation of the processing and storage of this data, which makes those systems responsible for the security and privacy of the user data.

In the interest of providing transparency to the owner of the data, the user whose location data is being stored should have access to this data and have full control over who can access it. One possible solution to enable transparency is the use of *personal data vaults* that provide control and transparency to the user.

2.5.2 Personal Data Vaults

Personal data vaults are secure storage spaces that are maintained by the person to whom the data belongs [135]. Third-party users, services or applications that want to store information about a person can request access to read or write to the personal data vault [43]. With many businesses such as Google relying on location data for data sourcing information about a user, personal data vaults ideally would force companies to switch to different business models [136]. Users using personal data vaults can still decide to share information with these companies in exchange for money or incentive features such as disk space.

In Figure 2.29, we can see the comparison between the current business scenario, where user data is stored and managed by each connected service individually, and



(a) Current business scenario where user data is stored and managed by each connected service individually.

(b) Personal data vault scenario where data is managed by the user and access is provided to the services.

Figure 2.29: Current business scenario and personal data vault scenario of user data (adapted from [136])

the personal data vault scenario, where data is managed by the user and access is provided to the services. This shift towards personal data vaults can lead to a more user-centric approach to data management, giving individuals more control over their data and potentially disrupting the existing data sourcing practices of companies relying on this data.

Personal data vaults require users to grant access to allow third-party services to access data within the personal data vault. User Access Control (UAC) is a method for users to manage access to their data and resources within a system [137]. In the context of location data privacy and security, User Access Control plays a crucial role in giving users the ability to control who can access their location data and for what purposes. By implementing User Access Control mechanisms, users can determine which applications or services have permission to access their location as well as the precision of the location, thereby enhancing their privacy and security. This control empowers users to make informed decisions about sharing their location data and ensures that their sensitive information is not misused.

One of the main benefits of personal data vaults is their ability to offer transparency on the data that is available within the data vault [138]. Users can access all their data and even review or revoke access rights to data that was previously granted to third-party services.

The Solid Project

In 2016, the W3C Solid Community Group led by Tim Berners-Lee introduced the *Social Linked Data (Solid)* [139] project, a platform for decentralised social

applications based on linked data (see Section 2.4.3). Solid stores a user's data and information in a secure data vault storage formerly called a *Pod* that is independent of the applications that create, edit or process the data. Users can choose the Pod provider and control the access rights of applications for certain subsets of their data vault. Stored data can range from binary media files, such as photos or documents, to structured linked data that is described by documented vocabularies and specifications.

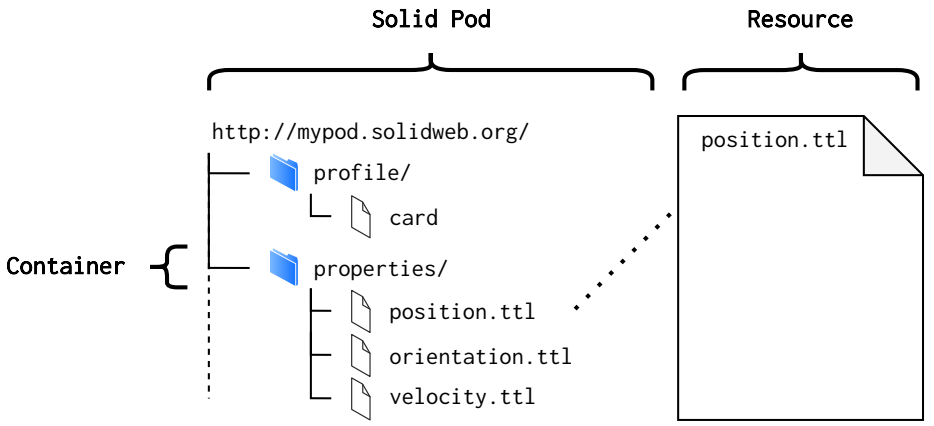


Figure 2.30: Basic Solid Pod structure

The Solid specification covers various aspects such as user authentication, vocabularies, and the protocol. Figure 2.30 illustrates the basic structure of data inside a Solid Pod. A user creates a Solid Pod at a specific URI (in the example `http://mypod.solidweb.org`). Inside a Solid Pod there are multiple *resources* that can contain linked data. To hierarchically structure resources, *containers* can be used, which can contain multiple individual resources. Each container and resource can have its own access control rights that define who can read or modify the data contained within those containers or resources.

The owner(s) of a Solid Pod are identified using a *web identifier* that uniquely defines the user or organisation who owns the data within the pod. These identifiers and their use cases for access control are detailed in this section.

A Web Identifier (WebID) is a URI that uniquely identifies an *agent* such as a person, organisation, project or application [140]. Solid Pods are identified by unique web identifiers that follow the conventions of the HTTP/HTTPS protocol. These identifiers serve as the location where the user's data is stored and managed within the Solid ecosystem. For example, the web identifier of the person "Maxim Van de Wynckel" (i.e., `https://maximvdw.be/profile/card#me`) points to the profile location of his Solid Pod.

One of the main features of Solid is its ability to provide access control to users or *agents*. Access rights can be configured per container or resource and can provide read, write, append and control rights for other users and groups. Users are identified by their WebID, while groups are a collection of users with their WebID.

- **Read**: Users or groups with read access can read all triples within a resource or container. In its current form, access rights cannot be specified for individual triples within a resource.
- **Write**: Having write access to a resource or container enables the user or group to edit and delete triples.
- **Append**: If we want a user to only be able to append or add new information, the append permission can be granted that allows the addition of triples but not the deletion.
- **Control**: Finally, control rights enable users to modify the access control rights for resources or containers.

In Solid, these access control rights are often grouped in roles such as poster, editor, visitor, owner and submitter. To authenticate applications to access a Solid storage, Solid uses open authentication [141], which is an authentication technique that sends an HTTP request to the Solid provider. After this request, the user is prompted to sign in, after which the Solid provider will send a request to the system that requested authentication. This final request contains an access token and an optional refresh token that is valid for a certain amount of time. These tokens can be used to perform actions on the Solid personal data vault.

Solid applications are web or mobile applications that are developed to interact with a user's Solid Pod. These applications can request permission from the user to access specific data or perform actions on their behalf within the Pod. Solid applications use the data stored in the Pod to provide personalised services to the user. For example, a Solid calendar application might access the user's calendar data stored in the Pod to display upcoming events and reminders. In our use case, a Solid application could be a positioning system or application that uses stored location data to visualise this to the user.

Blockchain-based Data Vaults

Blockchain is a digital ledger that is distributed among multiple smaller nodes in a network. Blockchain technology ensures the immutability and transparency of transactions by storing data in a decentralised and secure manner [142]. Unlike the Solid project, blockchain enables users and third-party services to verify the integrity of information. While Solid provides full control to users over the data that is stored within their Pod, a blockchain-based personal data vault keeps a record of changes and may optionally require that each change must be approved

by a majority of nodes [143] to be valid. This makes blockchain more reliable for storing personal information that requires trust, such as certifications, contracts or other personal data that may need to be shared with third-party services.

In Solid, this level of trust currently does not exist. However, when needed, a blockchain layer can be added on top of Solid or any other personal data vault to enhance security and trust in the data stored within the vault. By incorporating blockchain technology into personal data vaults, users and third-party applications can have a higher level of assurance that their data is immutable and verifiable.

2.5.3 Regulations

Regulations such as the General Data Protection Regulation (GDPR) [9] exist to protect the privacy and personal data of individuals within the European Union. The GDPR imposes strict regulations on collecting, storing, processing, and sharing personal data. Any application that deals with interoperable data, or location data in general, must adhere to the guidelines set by the GDPR to ensure the privacy and security of users' information. Another part of the GDPR is the right to *data portability*, which grants users the right to move their data to another provider or at least retrieve all the data that is kept. When answering our research questions, we will take this into account to design an interoperable positioning system that enables this data portability.

The Data Governance Act (DGA) [10] provides a regulation framework for enabling data sharing between different public sectors. Despite having a focus on the public sector, it also has an impact on the private sector, such as companies generating location data of users. Part of the DGA is to enable interaction and data sharing between the public and private sectors. With the increasing development of *smart cities*, it is not unthinkable that these smart cities may deploy indoor positioning systems for various use cases [144].

Nowadays, we rely on big players to keep track of our current location. These companies have access to a vast amount of personal location data, which raises concerns about privacy and data security. Companies such as Google used to track the locations of users in their *Timeline* product. However, to adhere to regulations, Google started to move the data from their cloud storage to local devices the user owns starting from 2024[4,5]. While this ensures they are not liable for this data, it also limits the availability and accessibility of the data by other services or even other devices. Furthermore, the processed information is still collected by Google according to their privacy policy [7].

[4] https://support.google.com/maps/answer/14169818?hl=en
[5] https://www.nytimes.com/interactive/2019/04/13/us/google-location-tracking-police.html

While we understand *accessibility* of interoperable applications to be the ability to access data from other services, this accessibility does not have to be public. This data can be encrypted or anonymised before being shared between applications to ensure the privacy of users and their data. If data sharing is required for the functionality of the application, access rights can be configured between applications. However, interoperability dictates that applications should be aware of the existence of the data — or the possibility of its existence and the steps that should be taken to access this data when available.

2.6 Discovering Positioning Systems and Services

Having a positioning system or location-based service (LBS) that can provide application interfaces and web services that provide location data is a useful first step towards interoperability. However, if users or applications are not aware of their existence, these services will remain unused and undiscovered. Discovering services and sensors is the act of finding and retrieving knowledge without prior knowledge of their availability. It is a well-known problem in the domain of internet of things (IoT) where applications and home automation services discover devices.

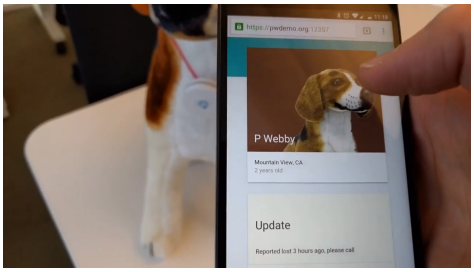
On the Web, in database systems and even in physical data stores, the discovery of data is often achieved by querying and indexing data. In the early days of the World Wide Web, the limited number of available websites was indexed by hand on centralised websites[6]. This created a *web of information* that could be traversed to find what you were looking for. As the number of websites increased, search engines that indexed all the available websites became essential for users to navigate the vast amount of data available online. Nowadays, when you want to visit a website to find information, you either know of its existence (i.e., its URL), you use a search engine that indexed the website beforehand or you happen to stumble upon a page of the website because it was referenced by another website (i.e., following links).

In the early 2000s, Hewlett Packard's CoolTown™ project [145] introduced the semantic description of people, places and objects. The *semantic locations* introduced as part of HP CoolTown were represented as URIs that were encoded within barcodes, physical locations using a trusted service or CoolTown beacons as shown in Figure 2.31. These CoolTown beacons were battery-powered infrared transmitters that could broadcast a uniform resource identifier (URI) pointing to an XML resource with semantic data every three seconds [146].

[6] https://www.w3.org/History/19921103-hypertext/hypertext/DataSources/WWW/Servers.html

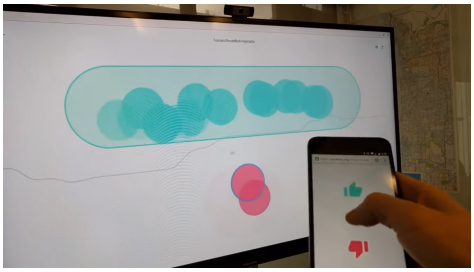
Figure 2.31: Hewlett Packard CoolTown project beacon [145]

Various related works focused on semantic location-based services [147, 148, 149] with a primary focus on navigation and environment description. However, they do not offer a method to advertise the availability of this location-based service in a physical environment or details on how applications should determine a location within these environments. Due to this limitation, applications require prior knowledge to discover environments and smart devices. Mathew et al [150] discussed several methods such as application interfaces, knowledge servers or open ontologies. Still, these methods only support the discovery of semantic descriptors of devices on the Web rather than in the physical world.



(a) Beacon attached to a dog linking to a web page with information about the owner[a].

[a] `https://youtu.be/-Y77cUI_z30`



(b) Beacon attached to a voting screen linking to an interactive web page to vote[a].

[a] `https://youtu.be/0SDJrRk3YC8`

Figure 2.32: Physical Web examples created by Google Ltd.

Google started the *Physical Web* project in 2014 to enable seamless interactions with physical *things* and locations [151, 152]. It was based on Eddystone-URL beacons to broadcast URLs for physical objects. Compatible smartphones would receive

a notification when they were near a physical thing broadcasting a URL, and the broadcasted URLs enabled direct user interaction without the need for additional applications. Eddystone-URL was based on the UriBeacon [153] specification, which was a project led by Scott Jenson, the UX designer who made the quote in the introduction of this dissertation. Reading through his blog posts around that time gives us insight into the vision behind the Physical Web project and the potential it held for seamless user experiences.

Figure 2.32 illustrates two examples of web pages advertised on the Physical Web. On the left, a beacon is attached to a dog that links to information about the owner. When the dog is lost, the owner can update the website to provide information about how to get in contact with the owner. On the right, a beacon advertises an interactive panel linking to a web page where users can vote. Despite the advancements in the Physical Web project, there were still challenges in the discovery of services and sensors in various environments. The focus has primarily been on broadcasting URLs for users to discover and interact with services, and not the interaction and discovery of two systems.

Similar to the discovery of data, the discovery of services is the process of finding relevant services for a particular goal. In a decentralised environment, the traditional methods of discovering services may not be applicable. Centralised discovery services, such as online directories or registries, may not be feasible or effective in a decentralised system where there is no central authority.

Conceptual Situation Spaces is a method that helps in finding semantic web services by matching those services with real-world contextual information and characteristics [154]. This is a method of context-aware data discovery [155] where a service is described with a semantic layer of information. While this enables the querying, reasoning and understanding of which services are linked to a physical environment, it does not solve our issue of decentralisation.

In other related work [26], two main types of data and service discovery are considered: global discovery and local discovery. Local discovery assumes the data can be discovered near the source, while global discovery assumes the data can be discovered across a wider network or system. In the context of discovering services in a decentralised environment, global discovery may involve a more distributed approach where services can be discovered across multiple nodes or devices.

2.6.1 Global Discovery

Global service discovery is a context-aware service discovery that does not know the general location of the service [156]. This type of discovery can be closely compared with Domain Name System (DNS), which offers a way to discover the IP address of a URL on the Web. A DNS server offers a hierarchical discovery

method to resolve domain names. A local DNS server will attempt to resolve the name in a local network. In contrast, country-level or global DNS servers take over the resolution when the underlying layer does not know a particular URL. One of the common issues with such discovery methods is the requirement to know the server(s) that can resolve this information.

Many implementations such as Service Location Protocol (SLP), Universal Plug and Play (UPnP) or even Lightweight Directory Access Protocol (LDAP) exist that attempt to solve the issue of global service and data discovery [157]. These implementations provide a way for services to advertise their presence in a network, allowing clients to discover and interact with them. However, these solutions are often limited to specific types of networks or require prior knowledge of the network structure.

In a decentralised system with an open-world assumption, global service discovery becomes a challenging task due to the lack of a centralised authority or registry. Peer-to-peer networks, blockchain technologies, and distributed ledgers offer potential solutions for global data and service discovery in decentralised systems [158]. Peer-to-peer networks often make use of a Distributed Hash Table (DHT). This table provides key-value pairs with the key being the hash of the data and the value being a set of peers that offer this data. Such distributed discovery relies on one or more centralised registries as the initial point of contact for bootstrapping the network and discovering other peers.

With digital twins, we assume that the virtual object is digitally available and can be discovered using computational techniques such as querying, accessing a URL or network discovery protocols. However, when we want to discover digital twins that are not located on the local network or when no knowledge is available on the data, different techniques need to be used.

A Digital Twin Registry (DTR) is a centralised database that stores information about digital twins [159], including their attributes, relationships, and metadata. The centralised registry serves as a central repository for information about different digital twins. By accessing this registry, users and applications can easily discover the availability of digital twins.

AnyPlace is an open indoor information service [160] and indoor navigation service that allows users to crowdsource building information via a centralised server. AnyPlace offers tools to facilitate crowdsourcing from known and unknown data producers via their API. While the server is no longer active, the project source code is publicly available for building owners who want to host their own version.

With our aim for interoperability, centralising the DTR for all positioning systems and environments would defeat the purpose of achieving interoperability. Instead, we propose to decentralise a digital twin registry, enabling each positioning system to provide its own digital twins that it is responsible for. This approach ensures that

each system maintains control over its information while still enabling discovery by users and applications.

Fog Computing

Outdoor Universal Plug and Play (OUPnP) [161] is an outdoor UPnP protocol [162] that enables the discovery of services and points of interest through *fog computing*. Other than cloud computing, where data is processed on servers, fog computing works with an additional *local* layer that processes information. Only when the local layer is not able to process the data, it is sent to a cloud service.

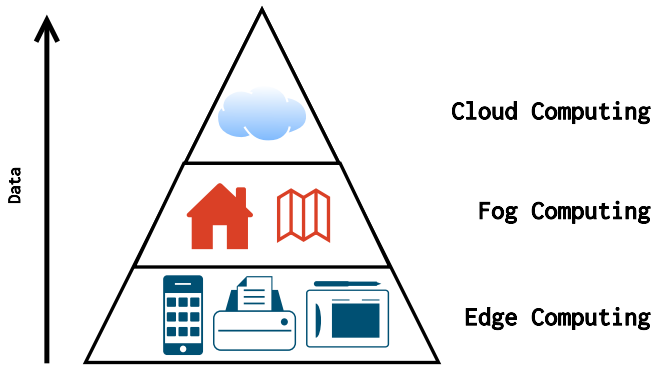


Figure 2.33: Cloud-, Fog- and Edge-computing

Fog computing has many similarities to edge-computing [163], which is a term used to describe the processing of information on small IoT devices in order to limit data communication and reduce latency. However, unlike processing the data locally on IoT devices, fog computing introduces a smaller layer in-between edge- and cloud computing, covering several devices belonging to the same spatial group. Figure 2.33 illustrates how data is generated by edge devices. Edge computing indicates that this data is processed "on the edge" at these devices, while fog and cloud computing introduce additional processing layers that require the transmission of this data to these layers.

Tiberkak et al. [161] introduce their OUPnP solution with local *fog* servers that cover a certain geospatial area. These servers can be used to discover devices within a geospatial area. When devices are not covered by the local fog server, they can communicate with a cloud service for further processing. This approach allows for more efficient discovery of physical things and digital twins within a specific geographical region, reducing the reliance on cloud services for every interaction.

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX poso: <http://purl.org/poso/>

SELECT ?system
WHERE {
  ?system a poso:PositioningSystem .
  ?system geo:hasGeometry ?geometry .
  FILTER(geo:sfWithin(
    ?geometry, "POLYGON((...))"^^geo:wktLiteral))
}
```

Listing 2.4: GeoSPARQL example query for positioning systems within a geographical boundary

Geospatial Querying

Geospatial querying is the retrieval of information from a database using geographical spatial boundaries as retrieval conditions [164]. When geospatial querying is performed on a global knowledge base, it can be used to query digital twins or positioning systems based on a rough geospatial boundary where the user is located. Unlike regular queries that compare information in the database with the query conditions, a geospatial query can provide the computation of distances and geospatial boundaries.

Databases such as MongoDB [165] or PostgreSQL [166] that support geospatial data already exist in positioning systems. However, these databases are often centralised and not accessible to third parties. This means that they require authentication to fetch or publish new information. If new geospatial data becomes available, it is up to the owner of the centralised database to add this information. To discover positioning systems or services, a decentralised solution needs to be available that would enable us to perform these types of geospatial queries without relying on a single database.

When we look at the Semantic Web as detailed in Section 2.4.3, GeoSPARQL is an existing vocabulary and extension to SPARQL endpoints (see Section 2.4.3) that enables geospatial querying over linked data [128, 167]. Apart from introducing a vocabulary to describe the location and boundaries of physical environments, this SPARQL extension introduces additional functions that can be used in queries to calculate if a position lies within a bounding box or to calculate the distance between two or more geographical coordinates.

Listing 2.4 demonstrates the GeoSPARQL extension that adds a vocabulary to describe geometries, while also adding SPARQL functions such as `sfWithin` to determine whether a geometry is within another geometry. In the example query, we query for all positioning systems (expressed with POSO [40]) within a specified

geometric boundary. While GeoSPARQL offers a method to query geospatial data, it does not handle the decentralisation aspect and has to be executed on a certain knowledge graph to retrieve information.

2.6.2 Local Discovery

Local discovery is a technique that enables applications to discover nearby services based on their proximity to these services. Other than global discovery, this type of discovery does not rely on the knowledge of servers that can aid in the discovery of services or data.

A QR code can be considered a type of local discovery where users register at a location while simultaneously providing information about the service that is being used to check in a user. An example of this type of discovery was used in one of our proof of concept applications towards decentralised positioning systems (see Section 6.4). In this proof of concept, we placed QR codes at the entrance of rooms. If these QR codes were scanned by the application that was created for this purpose, it would lead to a URL providing linked data about the environment. If another application scanned the QR code, it would redirect to the Web application that could then interpret the data in a similar way as the application.

As detailed earlier in this section, Google started the *Physical Web* project in 2014 to enable seamless interactions with physical *things* and locations [151, 152]. Part of the design philosophy of the physical web was the issue of IoT devices requiring proprietary applications for simple interactions. As this is not scalable with users not wanting to install applications, the physical web tried to act as a *proximity DNS* for finding nearby services. Interaction with these devices and services was always aimed towards user interaction through user web applications.

The physical web was implemented in the Android operating system as notifications that alerted the user when they were near an Eddystone-URL beacon. Due to an abuse of these notifications for *proximity marketing* [168], Google removed nearby notifications of Eddystone-URL beacons in 2018[7].

Furfari et al. [26] propose an Infrastructure-providing LBS (ILBS). They developed a *local* and *global* search for finding location-based services. A local search uses Eddystone-URL beacons, similar to the Physical Web, to discover nearby positioning systems. Applications that adhere to this local discovery will be able to retrieve meta information from these URLs about these location-based services through JSON data. On the other hand, global discovery requires a centralised server for querying information about various location-based services.

[7] https://android-developers.googleblog.com/2018/10/discontinuing-support-for-android.html

2.6.3 Open World Assumption

"Querying" is a concept used in informatics to describe the retrieval of information from a database or system through a set of constraints. It is a method of *discovering* information since the exact source or location of the data within the database is unknown. In a closed-world assumption, we assume that if the data cannot be found in the database that we are querying, and the query returns no results, then the information does not exist.

In an open-world assumption, we assume that the information we are querying for may indeed not be available in the current knowledge base, but could still exist outside the knowledge base. This is in contrast to a closed world assumption, where information not explicitly stated to be true is considered false. In the context of the discovery of positioning systems, we must also consider an open-world assumption. Just because we do not know of the existence of such a system, we cannot assume that there is no other knowledge base that contains information about it.

With the solution that we propose in this dissertation, we keep this assumption in mind. Interoperability between data sources will be as important as the interoperability of the data itself. Allowing data sources to connect and share information will enhance the expansion of knowledge that can be queried. Furthermore, the discovery method that we will design will have to work with incomplete data and the assumption that, while we are striving towards a single application that can interact with every positioning system, this will not always be achievable.

There's no sense in being precise when you don't even know what you're talking about.

– John von Neumann

Chapter 3

An Open-Source Hybrid Positioning System

Once we are using a single sensor that can determine a location or position, we consider the sensor to be part of a *positioning system*. A single sensor can often provide inaccurate results and may even be unable to produce an output in certain environmental conditions.

Most modern positioning systems, including the Global Positioning System (GPS), combine various technologies to compute a more accurate position. On a high level, hybrid positioning systems use the independent computed data from two or more technologies and combine it using high-level sensor fusion. This can range from a simple weighted average depending on which technology is the most accurate — to a more elaborate reasoning of the data based on the current conditions.

On a low level, fusion algorithms merge the raw sensor data to minimise the noise and errors of the data. A simple example of such a fusion algorithm is a complementary filter. It is commonly employed in sensor fusion applications, such as in navigation systems or robotics, to improve the accuracy and reliability of measurements. In the context of dead reckoning where we use acceleration to determine the drift in position, let us consider a scenario where you have two sensors measuring acceleration: one is an accelerometer, which is accurate in the short term but prone to drift over time, and the other is a gyroscope, which is good for measuring short-term changes but susceptible to noise. A complementary filter can be used to combine the strengths of both sensors. The low-level complementary filter combines these two measurements in a way that preserves the accuracy of the gyroscope in the short term and the stability of the accelerometer in the long term.

In addition, with indoor positioning techniques sometimes failing to determine a position in certain areas [33], the quote by mathematician John von Neumann (1903–1957) that *"There's no sense in being precise when you don't even*

know what you're talking about" appropriately applies to these systems. Hybrid positioning systems offer a solution by first establishing a reliable, if less precise, estimate of location before attempting to refine it further. In many cases, it is more effective to prioritise certainty in location, even if the updates are slower, rather than striving for high precision without a stable foundation. After all, precise tracking is meaningless if the system cannot first ensure the general accuracy of the location.

Interoperability of positioning systems, as defined as part of our main research question, cannot be achieved by semantically representing a single use case of a positioning system implementation. To clearly define how a positioning system operates and computes data, we need a generalised method to represent such a system and its data. As introduced in Definition 4, a hybrid or integrated positioning system combines multiple technologies and algorithms through sensor fusion. This makes a hybrid system generic enough to account for a multitude of variations, setups and scenarios.

In this chapter, we propose our solution to represent a general positioning system, or more specifically, a hybrid positioning system as defined in research question RQ1. The foundation of this generalised solution relies on the OpenHPS framework that was first released in 2020 and peer-reviewed at the Indoor Positioning and Indoor Navigation (IPIN) conference in 2021. OpenHPS is an open-source hybrid positioning system framework written in TypeScript[1] that allows developers to create and prototype positioning systems using a graph-based stream processing approach. We presented the framework at the IPIN conference in 2021, the Free and Open source Software Developers' European Meeting (FOSDEM) in 2022 and the Belgian JavaScript Conference (BeJS) in 2023. All prototypes and solutions related to this dissertation are built with the framework to demonstrate its effectiveness in generalising a variety of different use cases and processed data.

With this framework, we aim to target developers who wish to design a wide range of positioning systems and know the algorithms needed to achieve this. We also target researchers who want to design their own positioning techniques, without having to start from scratch. Future work may target a wider audience, such as non-technical end users or developers who want to implement systems with minimal development effort.

3.1 Methodology

The conceptual design of OpenHPS began in 2019 alongside the initiation of this research. Originally, we wanted to tackle the interoperability of indoor positioning systems by designing a framework that could become a new standardisation within the industry. However, we quickly recognised that this approach was too ambitious

[1] `https://www.typescriptlang.org`

and decided to narrow our focus to designing a data structure and processing pipeline that could represent a wide range of positioning systems. By doing so, we could offer a *positioning model* or representation that can be applied to different frameworks and future techniques.

Our methodology for designing OpenHPS draws inspiration from software architecture principles such as reusability, modularity and loose coupling, particularly from The Open Group Architecture Framework (TOGAF) [169]. We mainly based ourselves on the data and processing requirements of various positioning techniques and common sensors used to achieve this type of positioning. The main focus of an indoor positioning system framework is to handle location data, orientation data, velocity or acceleration, sensor fusion and the accuracy of the data. This type of information can always be generalised in a hybrid positioning system to some extent. Our requirements, outlined in Section 3.2, are therefore based on the requirements of a generalised hybrid positioning system.

To validate our design decisions, we open-sourced the software to obtain continuous feedback. OpenHPS was used for over five years in an exercise session of the course *Next Generation User Interfaces* by Master students of the Computer Science programme. In addition, the geospatial functionalities were used in exercise sessions for the course *Web Technologies* by Computer Science Bachelor students. We used the feedback and observations from these exercise sessions to improve the framework's convention over configuration and also to improve its documentation. During the five years of usage, we gained insights from industry players and have slowly started to apply these changes to the framework itself.

The primary validation of our design decisions lies within its data structure. We deliberately split the framework into multiple modules, each with its repository. This ensures that each module is modular and loosely coupled with other modules. We created several mobile applications, proof of concept positioning systems and experiments to validate that different use cases and data requirements were feasible within our framework. Our main proof of concept applications developed with OpenHPS can be found in Chapter 6.

3.2 Requirements

Based on existing positioning methods and algorithms discussed in Section 2.3, the following framework requirements have been derived. When defining these actors and requirements, we focused on functional requirements for representing a hybrid positioning system and the data it produces and consumes. We start by specifying the actors of our system and motivating the use of a processing network where each node of the graph topology might represent one of these actors.

Our system actors were defined at the beginning of the project through an analysis of existing positioning systems and frameworks. The terminology we used to name these actors is based on their goal. In Section 4.2, we validated these actors by semantically assigning each actor to part of a system. Finally, in Section 7.3, we settled on the final actors in our integrated solution.

The initial requirements, presented in our technical report of 2020 [32], were based on the actors and drafted based on our investigation of existing work, validated and discussed with group members of our research lab. In 2021, these requirements were peer reviewed and further refined based on feedback received at the IPIN conference.

During the development, some requirements were added or modified based on obstacles or shortcomings we encountered while implementing or testing the framework. OpenHPS consists of more than 400 unit and integration tests across all modules[2] that ensure that the framework satisfies these functional and non-functional requirements during its 5-year development lifecycle. These tests range from integration tests of third-party libraries, performance tests to ensure that workers are consistent, and functional unit tests. Tests were automatically performed for each commit and automatically propagated to all modules in the first few years of development to ensure consistency across modules. The complete framework, including modules enabling interoperability, discoverability, and unit tests, was peer reviewed for the Journal of Open Source Software in 2025 [34].

3.2.1 System Actors

After investigating different existing positioning techniques and systems defined by the ISO 18305:2016 [22], ISO 19116:2019 standard [46], we defined four actors in positioning systems, which we mapped in OpenHPS:

- **Tracked actor**: This is an actor that can be tracked during the online positioning stage. In indoor positioning, it might represent the user or object that is being tracked in an environment. Data that is processed in the network should have a reliable type and content. We process `DataObject`s encapsulated in `DataFrame`s (see Section 3.4.6), providing a defined scope on how generic parts of our network should handle information.

  A tracked actor can implicitly track another actor as well. In the case of a smartphone that is tracked by an external system, one might track the smartphone as a method to track the user who carries the phone.

- **Tracking actor**: This actor is responsible for tracking a tracked actor. It can be identical to the tracked actor (e.g., a user's smartphone) or some separate technology, such as a camera tracking the movement of a user. A tracking

[2] `https://github.com/OpenHPS/openhps-core/tree/master/test/specs`

actor has the highest priority in our processing model, and slow consumers or computing actors must not result in outdated sensor information. Rather, developers should be allowed to control what happens with any potential overflow of information that cannot be processed timely.

- **Calibration actor**: Some positioning methods require calibration or setup before they can be used. Unlike the tracked actor, the purpose of a calibration actor is to train and calibrate how a tracking actor will be used during the system's online stage. While we can assume that in some cases the calibration actor is the tracking actor, this is not always the case.

  To provide an example, a technician who uses a smartphone to set up an indoor positioning system is considered a calibration actor. This calibration actor will not utilise sensor data to obtain a location but will instead use this data for the set-up or training of the system.

- **Computing actor**: The computing actor is responsible for providing the final position output. It combines the data generated by one or multiple tracking actors and processes the data by using specific positioning algorithms to provide the absolute position of tracked actors.

  A smartphone that calculates its location locally on the phone is a calibration actor. If the smartphone itself produces information that can aid in the tracking, it is also a tracking actor. A smartphone that calculates its location

These four actors represent the four main components within OpenHPS. By distinguishing between the *tracked* and *tracking* actor, the system can support the tracking of persons or objects that do not actively participate in the positioning process. Note that our four actors can also be mapped to layers and design principles in the location stack [170]. Furthermore, computing actors represent an important component of the framework, as they are responsible for the processing and fusion of sensor data from multiple sensors. Based on these actors, we have defined the data structure and processing flow within OpenHPS to offer a general solution that can be applied to any type of positioning system.

3.2.2 Functional Requirements

In the following, we list the minimal functional requirements for our OpenHPS framework. These requirements are based on the actors as well as a broad range of positioning techniques such as fingerprinting, dead reckoning, and satellite-based positioning — as well as applications and other frameworks such as WebXR [171] or OpenVSLAM [50], and the requirements of those frameworks.

- **Offline stage positioning**: Our framework should support the collection, calibration and set-up of a positioning system. Results from this offline stage can be used in the online stage.

- **Online stage positioning**: To perform hybrid positioning or sensor fusion, multiple (processed) sources need to be combined by using different algorithms. Our framework should support the processing of this data, as well as access to previously stored information such as the calibrated data from the offline stage.
- **Third-party frameworks**: Our framework needs to support third-party high-level positioning systems. These external systems might provide their own calculated position of a tracked actor that needs to be fused with the position determined by our framework. In addition, the identification of this tracked actor might differ between frameworks.
- **Environment mapping**: With the requirement to support positioning methods such as SLAM and VSLAM, the system not only offers the possibility to output an absolute position but might also create an environment map. Our solution should be capable of handling, storing and using this map to its advantage.
- **Decentralisation**: Our positioning framework should be able to combine the four different actors introduced in the previous section based on remote hardware. This requires the framework to work decentralised without requiring any centralised sensor fusion, which can be achieved by allowing multiple computing actors to work independently. However, developers should still be given the option to centralise certain parts of the system if needed.
- **Monotonicity**: Partial information from a source should result in a partial output. In the context of a positioning system, this means that a computing actor does not need the sensor data of all positioning methods to determine a position. This requirement also helps in the decentralisation and parallelisation of the framework.
- **Serialisability**: All data processed and used by positioning systems created with our framework should be serialisable and deserialisable to other data formats. This ensures that the data produced by one positioning system can easily be transformed to another positioning system, which requires a different data format.

3.2.3 Non-functional Requirements

The following non-functional requirements contributed to the final decision about the software language used for OpenHPS. These non-functional requirements are primarily based on common industry practices regarding positioning systems.

- **Availability**: Our solution has to be available on various platforms ranging from servers to embedded systems, also supporting the decentralisation functional requirement. Having the platform available in multiple environments

will increase the system's accessibility and usability. Furthermore, if the system were to only work on one particular platform (i.e., a smartphone), it would be considered too domain-specific to one set of use cases.

- **Performance and latency**: Throughput is an important criterion when processing streaming data. Input data, such as video and audio streams, needs to be processed in real-time. The latency also indicates how long it takes for data to be used in computations. As our goal is to achieve an accurate current position, outdated sensor data is not relevant.

  While performance is important for the correct functioning of a positioning system, our primary focus, as indicated by our functional requirements, is on the data representation.

- **Modularity**: The framework should be modular with both a low-level API and modules that can be added and removed based on the available sensors and concrete use cases. Developers should remain in control of the types of algorithms and the flow of data from producer to consumer. Modularity is one of the main non-functional requirements of our framework, as we want to provide an extensible solution that can align our framework with future use cases that we did not foresee in our initial design.

- **Configurability**: The framework should allow developers to easily configure its components, data flow, and execution logic based on specific use cases, without modifying the core codebase. This includes support for declarative or programmatic configuration of sensors, algorithms, and deployment options.

When designing our framework, we also considered where our four actors would be implemented. Positioning systems are typically implemented in a distributed fashion, with different components running on various devices. The tracking actor, for example, might be implemented on a smartphone or a camera system, while the computing actor could be implemented on a centralised server or edge devices. Our decision to use TypeScript was based on its versatility to be used in server-side, web, mobile and embedded devices. This enables us not to limit our framework to specific platforms, which would negate the generalisation of such a framework.

One of the main challenges in designing a framework for real-time processing of data is ensuring the timely delivery and processing of sensor data. Meanwhile, as stated in our requirements, we wanted to design a modular framework that can be deployed on various platforms. The main problem we wanted to solve in this work is the lack of interoperability between (indoor) positioning systems (Problem 1). Within this problem, the main focus is on generality to help with the representation of the system. Therefore, our framework is data-centric in design [172]. We optimise performance in our framework to the extent that we can validate our data-centric approach with viable proof of concepts.

3.3 Architecture

Based on our requirements, we designed OpenHPS as a processing network that pushes or pulls data along a set of modular nodes that manipulate this data. Each node computes more information and passes it along in the network.

The data structure that is transmitted through our processing network will be detailed in Section 3.4. In general, it consists of an envelope that contains *objects* of information. To keep persistence between envelopes, *services* are added to the processing network that stores processed information when it reaches the end of the pipeline and retrieves previously stored information from these services when an envelope is created.

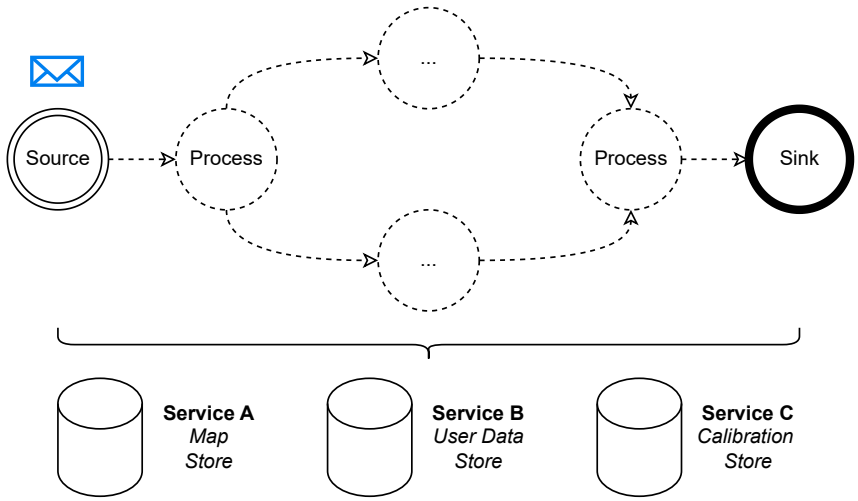


Figure 3.1: Example positioning system architecture in OpenHPS

Figure 3.1 illustrates the architecture of our OpenHPS framework. A graph of nodes is created to create a processing flow [173] from source to sink. A source node provides data, such as raw sensor information and processes this information step by step in each processing node.

The example positioning model in Figure 3.1 also shows three services, one for storing geospatial data, one of storing user data and finally a service for storing calibration data created during the set-up of the system. Every node in this graph has access to store or retrieve information from these services.

3.4 Data Structure

To address research questions RQ1.1 and RQ1.2, we have to define a common data structure that can be applied to a broad range of (hybrid) systems and technologies. We decided on two main data structures, (1) a *data object* that can represent any spatial data, whether it is a person, a smartphone, a room or a sensor and (2) a *data frame* which acts as an envelope to contain multiple data objects. Unprocessed values, such as raw sensor data, will pertain to one or more of these data objects. We include this raw sensor data together with all relevant objects within a *data frame*. In this section, we go into more detail on these two data structures.

As a framework to design and develop positioning systems, the position data itself is an important aspect of the framework. Similar to other related work [2, 54], we provide an absolute and relative position.

3.4.1 Orientation

Indicating the orientation of an object depends on the context in which it is used. If one were to ask in what direction someone is looking, they might indicate a specific angle in degrees, such as 45 degrees relative to the north. However, as this only indicates an orientation in two dimensions, it may not be sufficient in three dimensions.

Orientations in OpenHPS are extensible by design, allowing developers to represent them using different mathematical representations. However, internally in nodes of official modules, they will always be converted to quaternions due to their mathematical stability. Orientations can be created from Euler angles, axis angles, rotation matrices, bearings or any other implementation that can be expressed to quaternions.

```
const orientation = Orientation.fromEuler({
    x: 0, y: 0, z: 45, order: 'ZYX', unit: AngleUnit.DEGREE
});
```

Listing 3.1: Creation of an `Orientation` from Euler angles

Listing 3.1 demonstrates how an orientation can be created from Euler angles. Developers can decide on the order of the Euler angles as well as the unit in which they provide the orientation.

3.4.2 Absolute Position

As detailed in Section 2.1, an absolute position is a position with a fixed coordinate in a specified frame of reference. A *position* is a snapshot of the current position,

orientation and velocity of a data object. This makes a position very similar to a *pose* of an object, without specifically relying on the terminology.

Our decision to include orientation and velocity in a position was based on the fact that velocity and orientation influence a position itself. An orientation is also dependent on the position and its reference frame. Similarly, a velocity indicates the momentum of an object in motion. This momentum is relative to the object's current position and orientation and is thus also dependent on the reference frame.

Multiple implementations of absolute positions are included in the core component of OpenHPS, ranging from Cartesian 2D and 3D positions to geographical positions. Each absolute position can contain the following information:

Timestamp: All observational data, like positions, have a timestamp associated that indicates when the position was recorded. Since a position is a snapshot of the current

Unit: Regardless of whether the absolute position is a 2D, 3D or geographical position, a unit is required that specifies the value of the coordinates. In the case of a geographical position, this *unit* is replaced by the Geodetic Coordinate System (GCS).

Accuracy: The accuracy of the position is expressed in two or three dimensions, depending on the specified type of position. It is not a required property, but it can be set by the processing algorithms to provide additional context to other processing or fusion algorithms.

Orientation: An orientation can be created from Euler angles, axis angles, quaternions or rotation matrices and indicates the orientation of the object.

Velocity: The velocity of an object consists of the linear and angular velocity. It indicates the momentum of the object at the given timestamp.

Reference space UID: An absolute position is a fixed position on a frame of reference. Reference spaces are considered a type of data object and are identified with a unique identifier. Each position object has the UID of the reference space that is used. This links to a data object that contains more information about the space, including but not limited to the transformation to a global reference space.

```
1 const position = new GeographicalPosition(50.8205, 4.3921, 53,
  ↪ GCS.WGS84)
2     .setOrientation(Orientation.fromBearing(180))
3     .setAccuracy(new Accuracy2D(5, 5, LengthUnit.METER));
```

Listing 3.2: Creation of an absolute position with orientation and accuracy

Listing 3.2 demonstrates how a geographical position object can be constructed with a specified latitude, longitude, height and GCS. Orientation and accuracy are also included in the creation of the absolute position object. The orientation is set using a bearing of 180 degrees, while the accuracy is set to a 2D accuracy of 5 metres for the latitude and longitude.

Each instance of an absolute position has several utilities available to manipulate or access the data. These utilities can range from calculating the distance between two absolute positions to transforming the position from one reference space to another. These utilities provide flexibility and convenience in working with absolute positions within the framework.

3.4.3 Relative Position

While an absolute position indicates a relative position in a predetermined reference space, a relative position is a position *relative to* another object. Relative positions are commonly used in tracking scenarios where the position of an object is constantly changing with respect to a reference point or another object. Similar to an absolute position, each relative position has a timestamp, accuracy and unit. Various types of relative positions exist within the framework:

`RelativeDistance`: When the distance to another object is known, there is a relative distance between one object and another.

`RelativeOrientation`: A relative orientation is an Euler, axis angle, quaternion or relative bearing to another object with respect to the orientation of the absolute position and orientation of the object.

`RelativeLinearVelocity` and `RelativeAngularVelocity`: The relative linear and angular velocity indicate the velocity of the other object with respect to the orientation of the absolute position and orientation of the project

`RelativeRSSI`: Included in the `@openhps/rf` module, the relative RSSI indicates the received signal strength of an RF transmitter. This relative RSSI can be converted to an estimated relative distance.

`Relative2DPosition` and `Relative3DPosition`: An object that is positioned relative to another object is located at a certain Cartesian position relative to the other object. In most cases, sensors are not able to output the 2D or 3D coordinate relative to the reference point. However, when enough information is present, the relative 2D position and 3D position can be calculated.

Multiple relative positions can exist relative to the same object. As an example, a relative distance and relative angle can be combined to pinpoint the exact position relative to another object.

3.4.4 Data Object

A data object represents anything relevant to the positioning system. It can be the tracked object, the tracking object or a landmark needed for the relative positioning. Data objects can have a spatial and hierarchical relationship to one another. Examples of a data object can be a person, a smartphone of this person and even individual sensors within a smartphone. Our decision to choose one data object for multiple types of spatial actors was made because an *object* can be considered a generic representation of multiple actors. A smartphone, for example, can be both a tracking actor and a computing actor. Each object contains the following main attributes:

Unique identifier: Data objects are uniquely identified, either by a supplied identifier or a random UID. Optionally, a developer can provide a more user-friendly display name. However, the storage of data objects always uses the identifier.

Absolute position: Data objects store their last known absolute position. The stored position is always relative to the global reference space introduced later in Section 3.4.5. The relevance of this last known position can be determined using its timestamp, and developers can request a transformed position in their own reference space. A position is always accompanied by the orientation of the data object. Data objects are not required to have an absolute position if it is not relevant for the positioning.

Relative positions: These are relative positions to other reference objects. Each object can have multiple types of positions relative to different objects. This allows a data object to have a relative distance, angle and velocity to the same object. Relative positions act as the spatial relation to objects, which can be fixed or dynamic.

Parent object: A data object can specify its parent. This can be useful for indicating that individual sensor objects belong to the same tracked actor or that objects are spatially *inside* another object. When the position of a smartphone is known, we can use this position to infer the position of the user who owns the phone.

Depending on what the data object represents, it can be extended to store the information needed for its representation. In Listing 3.3, we create a basic data object of a user who is uniquely identified by their e-mail address. During the creation of this object, we set the current position to a geographical coordinate.

```
1 const object = new DataObject("mvdewync@vub.be");
2 object.displayName = "Maxim Van de Wynckel";
3 object.setPosition(new GeographicalPosition(50.82075, 4.39234));
```

Listing 3.3: Creation of a `DataObject`

Developers can decide what they use as an identifier for an object. In the case of an authenticated user, it makes sense to identify the object as the user itself, while anonymous users can be identified with data objects using a hardware serial number or similar.

3.4.5 Reference Space

A reference space is a type of data object that represents a space which can be used for determining an absolute position. Using these reference spaces, absolute positions created in a different space can easily be identified and transformed into the global reference space created when building a model.

```
const refSpace = new ReferenceSpace(model.referenceSpace)
    .unit(LengthUnit.CENTIMETER)
    .translation(10, 10, 0)
    .scale(1, 1, 0)
    .rotation(0, 0, 0, AngleUnit.RADIANS);
```

Listing 3.4: Creation of a `ReferenceSpace`

Listing 3.4 shows the creation of a reference space relative to the global space represented by `model.referenceSpace`. This reference space has an origin offset. Absolute positions set when providing this reference space will automatically transform to the origin of the global space.

A reference space can transform the position, velocity and orientation by providing a *translation* of the position with an origin offset; *rotate* the position, orientation and angular velocity; *scale* the position and linear velocity; transform the *perspective* or inverse perspective of the position and finally *unit conversion* to convert the unit of the position a reference unit. Reference spaces can be created to model a wide range of scenarios:

- **Third-party positioning systems**: Frameworks such as the WebXR [171] API manage their origin and orientation based on the underlying hardware. The output of such third-party frameworks is high-level positions that should be aligned with the other positioning methods.
- **Sensor placement**: Developers can model a reference space for sensors that have a static offset or rotation (e.g., a motion sensor that is placed upside down).
- **Calibrated reference space**: Some sensors require a calibration (either automatic or by manual user input). A goal of OpenHPS is to easily persist this type of calibration.

- **Map storage**: As a data object, a reference space can be extended to store environment map information as outlined in our functional requirements.

In Listing 3.5, we set the current position of a data object to $(5, 5, 5)$ using the previously created reference space shown in Listing 3.4. Internally, the stored position of `myObject` will be the transformed position with coordinates $(-5, -5, 5)$.

```
1 myObject.setPosition(new Absolute3DPosition(5, 5, 5), refSpace);
```

Listing 3.5: Setting the object position in a reference space

As these spaces are data objects, they are uniquely identified and can have a parent object or space. This parent allows for abstract reference spaces such as *rooms*, *floors* and *buildings*. These types of abstractions allow us to use different positioning methods per floor that are stored in a global reference space representing a building.

In our `@openhps/geospatial`[3] module, we expanded upon this data type with the concept of a *symbolic space*. Symbolic spaces represent a high-level API extension of reference spaces aimed at indoor environments. They add the following capabilities on top of reference spaces:

- **Spatial hierarchy**: Spatial hierarchy is already supported in reference spaces using the parent identifier inherited from regular data objects. However, symbolic spaces add boundaries that can be used to indicate whether an object is inside a space.
- **Graph connectivity**: The ability to connect spaces such as rooms, hallways, staircases and floors to support navigation applications or improve position estimation.
- **Geocoding**: Symbolic spaces can be converted to an absolute position in the global reference space. Similarly, an absolute position can be converted to the most likely symbolic space through reverse geocoding.
- **Semantics**: Symbolic spaces also introduce semantics to reference specific types of spaces, allowing developers to define the purpose or use of a particular space.
- **GeoJSON export**: Spaces might be exported to GeoJSON [174] features, aiding the storage and query capabilities in MongoDB or other data services.

Symbolic spaces can be treated as *symbolic locations* [175, 176, 58] with the ability to be converted to absolute positions. By using the terminology of *symbolic* space

[3] `https://openhps.org/docs/geospatial/`

as opposed to other terminologies, we offer a general environment that can also apply to a reference space such as a *table*.

```
const building = new Building("PL9")
  .setBounds({
    topLeft: new GeographicalPosition(50.8203, 4.3922),
    width: 46.275, height: 37.27, rotation: -34.04
  });
const floor = new Floor("PL9.3")
  .setBuilding(building).setFloorNumber(3);
const office = new Room("PL9.3.58")
  .setFloor(floor).setBounds([
    new Absolute2DPosition(4.75, 31.25),
    new Absolute2DPosition(8.35, 37.02),
  ]);

const object = new DataObject("myuser");
// Set the position relative to the floor space
object.setPosition(
  new Absolute2DPosition(6.55, 34.135, LengthUnit.METER)
), floor);
// Get the position relative to the global reference space
office.getPosition(); // (lat: 50.8204, lng: 4.3922)
// Get the position relative to the floor
office.getPosition(floor); // (6.55, 34.135)
```

Listing 3.6: Symbolic space creation and usage

Listing 3.6 shows the creation and usage of three symbolic spaces. Boundaries can be specified through different methods. In this example, the boundaries of a building are defined via the top left corner, the width and height of the building in metres, as well as the building's orientation (angle) relative to grid north. For the boundaries of the floor and a specific office room, we provided two boundary points that create a rectangular symbolic space. Alternatively, polygonal shapes can be used to define the boundaries.

Figure 3.2 illustrates how the symbolic spaces created in Listing 3.6 show up on the map when exported to GeoJSON. OpenHPS also enables developers to parse symbolic spaces from GeoJSON data. However, apart from providing additional context and reference frame transformations, these symbolic spaces are currently not used by any positioning algorithms included within the framework.

3.4.6 Data Frame

Data that is pushed through the positioning model is represented within data frames, generated or pushed by a source node. They act as the *envelope* of data that is sent

Figure 3.2: GeoJSON representation of a floor with rooms

through a process network. In a way, they are momentary snapshots of information that are processed step by step. This encapsulation of information ensures that the origin of data can be determined through a collection of metadata. The data contained in these frames includes (but is not limited to) the following attributes:

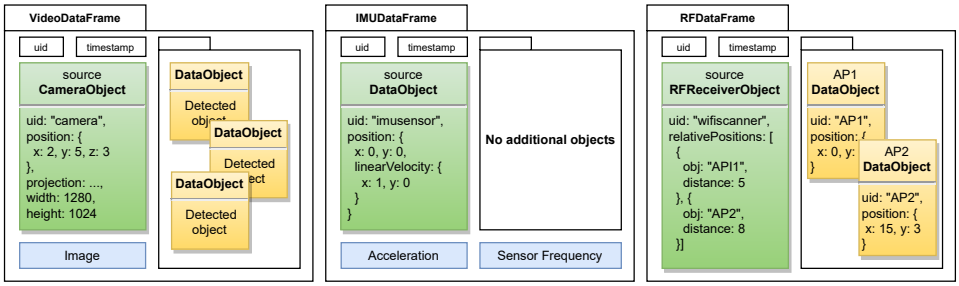


Figure 3.3: DataFrame examples

Unique identifier: Each frame generated by a source is uniquely identified. This ensures that frames which are being processed by multiple processing nodes in parallel can be merged at a later stage in the stream.

Timestamp: Required for determining when the data was created or obtained. When working with multiple sources that capture data of the same tracked actor, the timestamps will be used to merge the data frames. A timestamp is kept for the creation of each data frame by the source. This timestamp can also be used for time-based calculations, such as applying velocity to a position. Using this timestamp instead of the system time results in a more deterministic output.

Source data object: This is the data object that obtained the sensory data (e.g., the camera object or RF receiver). It is not always the object that is being tracked, but it can be required in order to determine the position of other objects. Similar to

the timestamp and identifier, the source data object can be used to specify certain criteria on how data frames or positions should be merged.

Data objects: Data objects include everything that is of relevance to the positioning (e.g., the tracking and tracked actor). This also includes reference spaces needed for the positioning as pointed out later in Section 3.4.5. By grouping the data objects in the same data frame, nodes do not have to access any services to get this relevant information.

Raw sensor data: Raw unprocessed sensor data that is captured by the source node is included within a data frame. As an example, in the case of a camera generating a stream of video frames, the individual frames are included within the data frame itself.

To demonstrate the content of data frames, Figure 3.3 depicts three situations where data is contained in frames. The first example shows a data frame created by a camera source. This camera object has a certain position and a projection matrix. Linked to the data frame is a single image (i.e., video frame) captured by this source. During the processing of the image, objects can be detected and added to this frame before being pushed further downstream. In the second example, we show data obtained by an accelerometer. The source object has a velocity and position, and the frame itself contains the current acceleration and sensor frequency. This information can be used by a processing node to add acceleration to the existing velocity. In our third and final example, we show a data frame created by a Wi-Fi scanner. The scanner (source) has two relative distances to access point (AP). The information, mainly the position of these access points, is included in the frame.

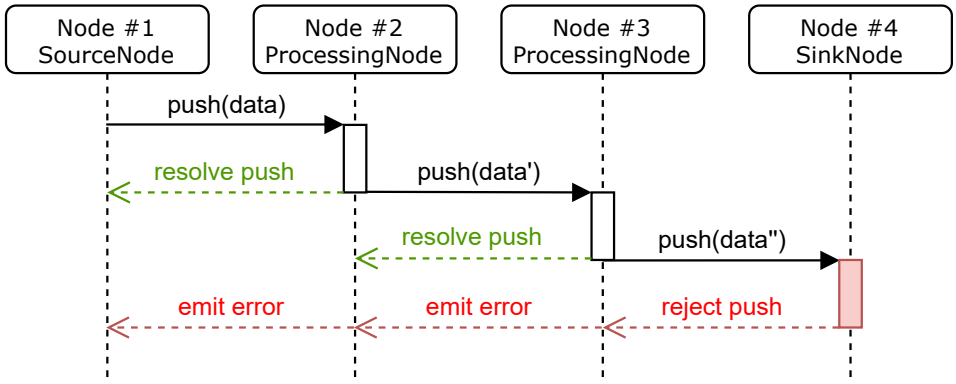


Figure 3.4: Error handling in `push()` request

3.5 Measurement Units

Unlike many other positioning frameworks that offer geographical positioning, OpenHPS aims to support a wide range of use cases ranging from small-scale to celestial positioning. To enable this wide range of use cases, we allow developers to provide a unit for all measurements within data frames, objects or sensor data. We enable this by adding a unit system consisting of a `Unit` and `DerivedUnit` object. A derived unit is *derived* from multiple units with a specific power and offset. Math.js [177] offers a similar unit system with the possibility to automatically evaluate and convert units by providing a textual formula. While this allows for the easy creation of derived units, it is not necessary for our framework and is more difficult to expand.

```
1  const second = new TimeUnit('second', {
2    baseName: 'time', // Unit for 'time'
3    aliases: ['s', 'sec', 'seconds'], // Alias for "second"
4    prefixes: 'decimal', // Supports decimal prefixes
5  });
6  const millisecond = second.specifier(UnitPrefix.MILLI);
7  const minute = new TimeUnit('minute', {
8    baseName: 'time', aliases: ['m', 'min', 'minutes'],
9    definitions: [{ magnitude: 60, unit: 's' }], // 60 x 1 sec
10 });
```

Listing 3.7: Unit creation

Listing 3.7 shows the creation of a base unit `second` for time. During its creation, the developer can specify aliases for the unit and similar to Math.js, a unit can have a set of unit prefixes. This allows using "millisecond, microsecond or nanosecond" without specifically creating individual units for these specifiers. Note that aliases can be provided to optionally allow the units to be converted to string evaluators of other mathematical modules.

When creating a new unit, the developer should specify the base unit. The time unit "minute" example in Listing 3.7 is done by creating a definition for converting minutes to seconds (using a magnitude of 60 for the existing time unit "second").

To use a unit that is derived from other base units, a `DerivedUnit` can be created as shown in Listing 3.8. The developer provides the name of the unit and adds the units that are contained in the derived unit (line 4) along with their magnitude. Variants on derived units can be created by *swapping* a unit (lines 5 and 8).

One of the design philosophies of our framework is not to force developers to use specific units when representing data. Depending on the type of positioning system, different units may be needed. For example, when working on a positioning

```
const radSecond = new DerivedUnit('radian per second', {
    baseName: 'angularvelocity',
    aliases: ['rad/s', 'radians per second'],
}).addUnit(AngleUnit.RADIAN, 1).addUnit(TimeUnit.SECOND, -1);
const degreeSecond = radSecond.swap([AngleUnit.DEGREE], {
    baseName: 'angularvelocity', name: 'degree per second',
    aliases: ['deg/s', 'degrees per second'] });
const degreeMinute = radSecond.swap(
    [AngleUnit.DEGREE, TimeUnit.MINUTE], {
    baseName: 'angularvelocity', name: 'degree per minute',
    aliases: ['deg/min', 'degrees per minute'] });
```

Listing 3.8: Derived unit creation

system for detecting objects on a table, different units are used than in a system for tracking cars on the road. Throughout the framework, every value can have its own unit. Processing nodes can normalise these values to the same unit to perform calculations.

3.6 Serialisability

In our non-functional requirements of Section 3.2.3, we indicated that part of our decision to choose TypeScript as the programming language for our framework was its ability to be used on different platforms. Positioning systems often consist of decentralised parts that each compute and process data independently. Therefore, it is crucial that all data, such as data objects and frames, can be communicated between these platforms.

All data represented within OpenHPS, whether it is a data object, frame or a single sensor measurement, is serialisable and deserialisable. OpenHPS offers a custom implementation of the TypedJSON library[4]. This library offers *decorators* to developers that can be added to classes and members to provide metadata about the serialisation and deserialisation of JSON.

Decorators in TypeScript are functions of classes, fields and methods that are executed when a class loads. In TypedJSON, these functions enrich the underlying *prototype* of a class with metadata about the fields it should serialise, and how this serialisation or deserialisation should be performed. Appendix A.3.1 illustrates a custom data object class with decorators for the class and members.

We have extended TypedJSON to better support polymorphism and also to facilitate the expansion of the serialiser and deserialiser logic in other modules. Addition-

[4] https://github.com/JohnWeisz/TypedJSON/

ally, we modified the decorators to allow developers to provide more metadata concerning the data types of fields. This additional metadata enables us to create *schemas* at runtime for databases or communication methods that require a fixed schema.

TypedJSON already provides serialisation and deserialisation of TypeScript objects. However, during the implementation of our requirements, we experienced difficulties in achieving a modular framework with extensible data. Our primary change to TypedJSON was to create an overarching layer that captures the creation of metadata using decorators. This provides us with the necessary control to extend data objects in different modules or to serialise to different data formats such as RDF.

```
{
  "createdTimestamp":1606501972983302,
  "uid":"8865727c-7c98-4a8d-a33c-506d2650e59d",
  "position":{
    "x":-4.0709, "y":55.5913, "unit":{ "name":"centimeter" },
    "timestamp":1606502001594449,
    "velocity":{
      "linear":{
        "x":-0.2760, "y":0.3606, "z":0.0132
      }, "angular":{
        "x":-3.9937, "y":0.2311, "z":-0.5070
      }
    },
    "orientation":{
      "x":-0.0975, "y":0.1538, "z":0.0426, "w":0.9823
    },
    "referenceSpaceUID":"5582d63d-c7af-4624-9fed-6ce0d9036f62",
    "accuracyUnit":{ "name":"meter" },
        "__type":"Absolute2DPosition"
  }, "relativePositions":[], "__type": "DataObject"
}
```

Listing 3.9: Serialised `DataObject`

Listing 3.9 shows a serialised data object. The main `DataObject` and position have a `__type` key that defines the object type. Definitions of a unit are not included in the serialisation and its complete name is used to indicate the unit. This means that a custom unit should be available in all processes that are required to deserialise the unit.

With this serialisability, we want to ensure that all data can be easily transferred between nodes, whether it is the transmission of data between two different systems or between processes on the same machine. The serialisability of data on the same

machine is used to enable the multithreading of positioning systems developed with OpenHPS (see Section 3.11).

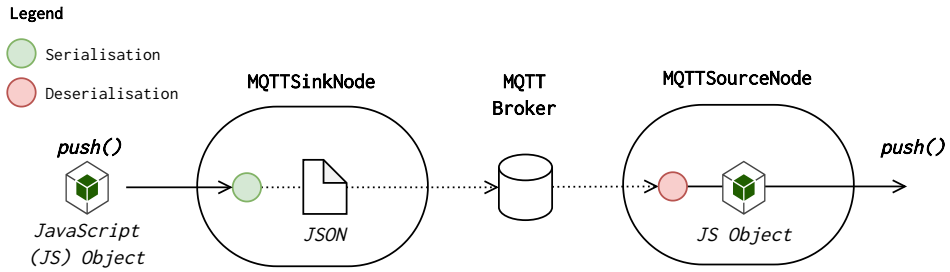


Figure 3.5: Transmission of data from a sink to a source using an MQTT broker

Internally in a graph, JavaScript objects of `DataFrames` are pushed from one node to the other. Figure 3.5 illustrates how data can be transferred from the sink of one graph to the source of another using Message Queuing Telemetry Transport (MQTT). On the left in the figure is a `MQTTSinkNode` that receives a JavaScript object. To transfer this object via MQTT, it is first serialised, after which it is sent to the MQTT server. On the right of the figure, an `MQTTSourceNode` actively subscribes to changes on the MQTT server. Whenever new data is available, this JSON data is deserialised before pushing it as a JavaScript object to the next node in the graph. The serialised JSON representation is used internally within the framework and is not interoperable by default. However, in Section 4.3, we expanded the serialisation of data to RDF and JSON for Linking Data (JSON-LD). This ensures that the data can be easily integrated with other systems that support RDF data.

3.7 Graph-based Stream Processing

To support the presented functional and non-functional requirements, we decided to build on a *stream-based positioning system* that takes various types of *input data* and processes this data to get the desired output. Data that is transmitted between nodes is encapsulated in so-called *data frames* that can contain sensory data as well as one or more *data objects* the sensor data applies to, and are described in detail in Section 3.4.

For the design of our process network, several existing stream- and layer-based frameworks such as Akka Streams [178], Apache Kafka [179] or TensorFlow [180] have been investigated. Since each node needs to be configured individually, the decision was made to investigate flow-based frameworks where each component of the stream network is added individually.

Unlike low-level data stream frameworks, OpenHPS focuses on data that is helpful for positioning. We offer a higher-level API for creating the network and data that is handled by the system. Concepts such as edges or ports that are often found in stream-based programming languages are abstracted and not directly accessible by developers. However, unlike other hybrid frameworks [75, 57], the stream processing is extensible enough to allow developers to modify the positioning methods along with the used algorithms.

We start by discussing our process network design that uses a graph topology similar to other stream frameworks. Next, we present the data frames, objects and positional data that are being handled by the network. Each node can be designed to accept both push and pull requests. Similar to reactive streams [181], push and pull actions are *promise*-based and can be executed asynchronously. If a node that receives a pull request cannot respond with a data frame itself, it will forward the pull request to its incoming node(s).

Different to a traditional pull that returns a *response*, we use the *push* terminology to indicate a response for a given pull. This behaviour and terminology are similar to Akka Streams [178], but unlike reactive streams, where data can only be provided when there is a demand, there is no back pressure built into the stream itself. Using the push terminology for a pull response removes the ambiguity of a response arriving after an already existing push in the pipeline. It also enforces the design goal of producers having the highest priority, even if a producer only generates information when requested.

A regular node has a unique identifier and push/pull functionality for data frames. Each node can have $0 \dots n$ inlets or outlets. Our system consists of the following three subtypes of the regular node:

A `SourceNode` provides a specific data type. This can either be a push or pull node that pushes data frames when they are available (e.g., a camera recording at a fixed frame rate) or creates a new data frame when the downstream node asks for it via a pull request (e.g., triggering a Bluetooth scan). The source node merges *data objects* in the data frame with those that were previously stored via *data services*. This merging behaviour prevents the need for feedback loops to gain knowledge on previously calculated positioning data.

`ProcessingNode` is a higher-level interface of a regular node. It provides an abstraction of the push and pull functionality to simplify the creation of a processing function of either data frames or individual data objects. This allows developers to focus on the processing aspect, rather than having to distinguish between push and pull actions. The processing node can be related as a *computing actor* as defined in our requirements.

An output node or `SinkNode` accepts a specific data type as an output frame. Unlike processing nodes, this type of node will not push data to other nodes. Upon

receiving a data frame, the data objects will be stored using a compatible data service. Once saved, an event is sent upstream to indicate that the processing of this frame and its contained objects is completed.

Extensions of these nodes, allowing for specific data flow shapes and common position processing nodes, are provided in our core component. Similar to existing streaming or pipelining frameworks, the graph can contain data flow shapes that manipulate the flow of data frames. Examples of such shapes include, but are not limited to, balance nodes, data frame chunking, debouncing and merging of data objects and their processed positions.

Processing nodes can perform any algorithm that is relevant for calculating a position or other related data. Modern positioning techniques often utilise deep neural networks to train and process information. We developed the `@openhps/tf` module that adds TensorFlow [180] models as services or individual nodes. A neural network in itself is a graph, which is abstracted in OpenHPS as a graph within a graph with some additional logic that transforms the input and output data to neurons in a neural network.

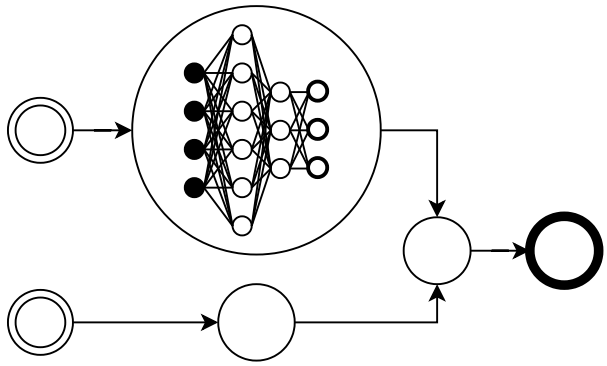


Figure 3.6: Illustration of a neural network within an OpenHPS processing node

Figure 3.6 illustrates this example by visualising a neural network with the processing node of the positioning model. Our `@openhps/tf` handles the wrapping of the neural network model to transform `DataFrame`'s and `DataObject`'s to input neurons that aim to process this input data to a result. This result is merged with the data frame before being pushed to the next node in the graph.

In Figure 3.7, data is being pushed by an *active source* node. Processing nodes will process the data and push the modified frame to their output nodes. Push and pull actions are promise-based and resolved whenever the node finishes processing the frame. This allows for non-blocking asynchronous requests. The resolved push promise (indicated in green) indicates that the processing of the push is finished. However, it does not provide knowledge on whether or not the frame is processed by the complete network. To indicate this, sink nodes that receive a frame will

emit a `completed` event that includes the data frame identifier and list of persisted object identifiers.

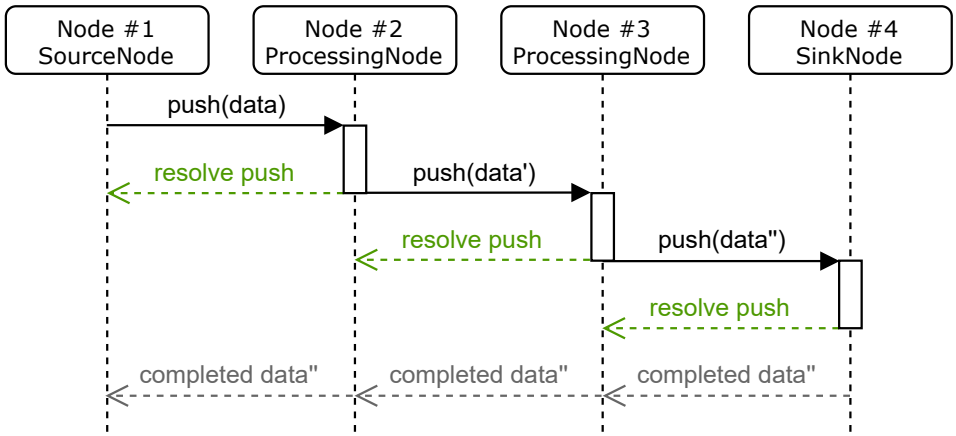


Figure 3.7: Data being pushed by a source node through the model

With the swim lane shown in Figure 3.8, the data is not automatically pushed by the source node. A downstream node, such as a sink, will send a `pull()` request to its input nodes. If these nodes cannot provide a frame of their own, the `pull()` request is forwarded to their respective input nodes. If the source has data available, a response to this pull is provided asynchronously. As mentioned at the beginning of this section, a `pull()` response will use the same invocation as a `push()`. In that case, the pull promise is resolved right after the source sends this push, as indicated by the blue resolve chain in Figure 3.8.

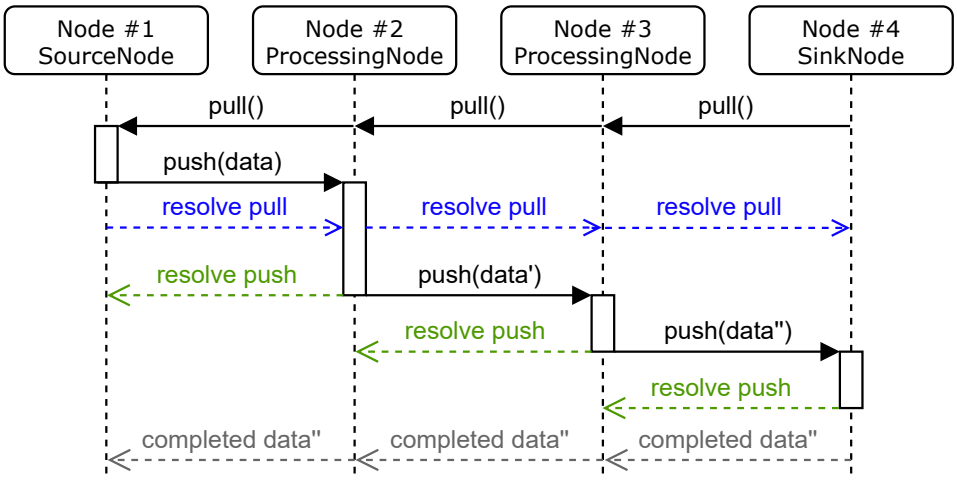


Figure 3.8: Data being pushed by a source node after receiving a pull request

As promises are resolved after the data frame is processed by a node, upstream nodes in the process chain cannot determine whether data has been processed successfully. Figure 3.4 shows a `push()` request that throws an error at the sink node (e.g., failure to store). An error event is triggered on a previous node(s). By default, these nodes will chain the error to upstream nodes. However, each node can act upon this error in its own implementation.

Nodes are implemented by developers on a high level of abstraction compared to other stream processing frameworks. Developers cannot push or pull from specific incoming or outgoing edges. Listing 3.10 shows two custom source nodes. The pull-based source node on lines 1 to 7 implements the `onPull()` function that is called whenever the source receives a `pull()` request. This function expects a promise of a data frame. Internally, the extended source node class will push this data frame as shown in Figure 3.8. With the push-based source (lines 9 to 22), the `onPull()` is unused. Instead, a timer is created that pushes a new data frame every 1000 milliseconds. A similar abstraction exists for sink nodes with the `onPush()` function.

```
1  export class PullBasedSource extends SourceNode<DataFrame> {
2    public onPull(): Promise<DataFrame> {
3      return new Promise((resolve) => {
4        resolve(new DataFrame(this.source));
5      });
6    }
7  }
8  export class PushBasedSource extends SourceNode<DataFrame> {
9    constructor(source: DataObject) {
10     super(source);
11     this.on('build', () => {
12       setInterval(this._generate.bind(this), 1000);
13     });
14   }
15   private _generate(): void {
16     this.push(new DataFrame(this.source))
17   }
18   public onPull(): Promise<DataFrame> {
19     return Promise.resolve(undefined);
20   }
21 }
```

Listing 3.10: Push- and pull-based `SourceNode` classes

Similar to sources and sinks, processing nodes are abstracted. Any `pull()` requests to these nodes are automatically forwarded to the incoming nodes, as these process

nodes do not generate new data frames. Developers are expected to implement a `process()` function manipulating a frame or individual objects within a frame.

3.8 Data and Processing Services

Each positioning model can have multiple services. A service can be accessed by all nodes in that model to perform certain general actions, ranging from communication services that handle the data between remote nodes, to data services that store data frames, objects or other relevant information.

In the given example of Figure 3.1, three services are added for the storage of map, user data and room information. In our implementation, sink nodes always store data objects contained in received data frames. However, every node can fetch or insert new data into available services. This persistence allows for the storage of landmark objects, similar to the JSR-179 specification [66]. At the same time, these services can be used as an interface to fetch the latest position without requiring a specific implementation in the sink.

A data service serialises and stores information. By default, our core API offers data services for storing the processed objects and their last known position. This can also be used as a persistent storage for landmarks used in the positioning. Additionally, node-specific data about `DataObject`s can be stored. This can be useful for intermediate calculations by noise filtering algorithms or sensor fusion techniques. Historical position data or trajectories of `DataObject`s may also be useful for utilising historical information to more accurately determine a new position. Drivers can be implemented for storing this information in specialised databases such as MobilityDB [182].

Normal services in our framework include, but are not limited to a *time service* that allows developers to synchronise the time between multiple machines, and a *worker service* that acts as a (remote) proxy for data services. More advanced services, such as a *fingerprinting service*, handle the processing of information between the online stage and offline stage of a positioning system. The `TimeService` enables basic clock synchronisation between (distributed) nodes. While this service currently does not tackle latency issues in a distributed context, it is extensible to support this in the future.

The positioning model can be created by using a builder pattern as illustrated in Listing 3.11. This builder creates the immutable properties of the model, including data services and the flow of data from source to sink. Models can have multiple flow shapes, each with one or more sources, processing nodes and sinks.

Listing 3.12 shows examples of how a service can be retrieved from the model. Nodes can retrieve a data service by providing either the class of an object, an

```
ModelBuilder.create()
  .addService(/* ... */)
  .addShape(GraphBuilder.create()
    .from(/* ... */).via(/* ... */).to(/* ... */))
  .build().then((model: Model) => { /* ... */ });
```

Listing 3.11: Creation of a positioning model using OpenHPS

object instance or the class name of the object. This allows the use of different data services for different types of `DataObjects`.

```
this.model.findDataService(DataObject); // Find by object class
const myObject = new DataObject("Maxim");
this.model.findDataService(myObject); // Find by object
this.model.findDataService("RFDataObject"); // Find by name
```

Listing 3.12: Retrieving a data service from a model

3.8.1 Querying

Data services support querying to retrieve specific data frames or objects. Querying data in OpenHPS uses the MongoDB syntax [166] for writing queries, which makes use of the *properties* in the data to select the appropriate data.

```
{ displayName: "Maxim" }
{ "position.x": { $gt: 10, $lt: 20 }}
{ $or: [{ displayName: "Beat" }, { displayName: "Maxim" }] }
```

Listing 3.13: MongoDB query syntax examples

Listing 3.13 illustrates three examples of MongoDB queries. In the first example, a data object is selected based on its display name. In the second example, data objects are selected where the X-value of the position is greater than 10 and less than 20. Finally, the third example selects data objects that either have the value “Maxim” or “Beat” for the display name.

This query syntax is used throughout the whole framework regardless of whether MongoDB is used as the driver for storing the data. The default in-memory driver of OpenHPS will interpret the MongoDB query syntax to retrieve the appropriate objects in memory, while other drivers, such as our linked data module (`@openhps/rdf`), which is further detailed in Section 4.3, transform the queries to other query languages such as SPARQL.

3.8.2 Post-processing Data

Data services can be used to persist information such as data frames and objects. The data of these objects is often processed by nodes within the processing graph. However, in some cases, the data processing requires a large set of other relevant information that is not present within a data frame. In Section 2.3.2 of our background chapter, we detailed how fingerprinting uses an online and offline stage. During the offline stage, fingerprints are collected and processed, while the online stage uses these processed fingerprints to determine a position. While our primary goal is to *store* data, this data requires processing using the complete set of information (i.e., fingerprints).

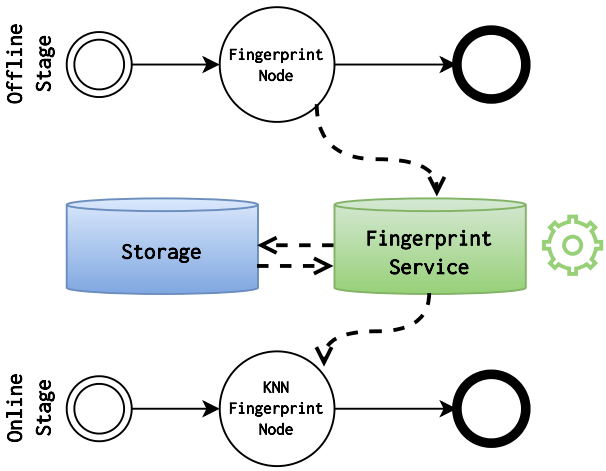


Figure 3.9: Fingerprinting service used in the offline and online stage

To solve this issue, a processing service is introduced that processes the data before storing it using an appropriate data service. Figure 3.9 illustrates how two processing graphs each have a fingerprint node. In the offline stage, this node extracts the *features* from the data frame (e.g., RSSI readings) along with the recorded location where these features were captured. These features are sent to the `FingerprintService`, which collects the data and processes it into a fingerprint database. During the online stage, the same service is used by a specific fingerprint algorithm to retrieve the processed fingerprints. An elaborate example of this fingerprinting service is available in Section 6.3.

3.8.3 User Actions

User actions are often steps that cannot be easily integrated into the existing *unmanaged* pipeline of sensor data. In OpenHPS, individual procedures or nodes do not affect the speed at which the source node provides data. Often, users need to calibrate sensors by performing actions that could block the flow of data. In our

unmanaged pipeline, this would drastically slow down the processing unless the developer implements nodes to circumvent this blockage.

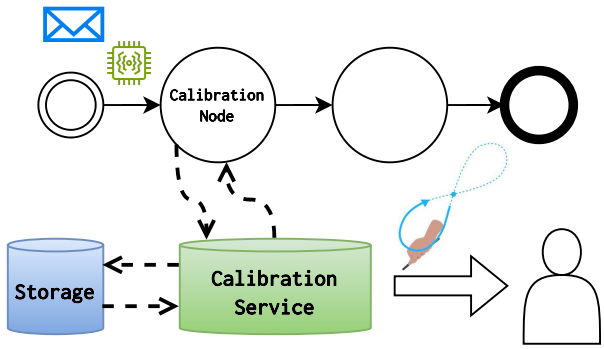


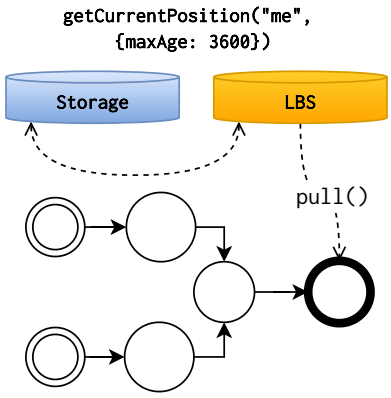
Figure 3.10: User action service to calibrate a sensor

Our solution to these user actions, as illustrated in Figure 3.10, is a *calibration node* and *calibration service* that serve the purpose of both the offline stage and online stage in a positioning system. When calibration data for a particular sensor (i.e., `SensorDataObject`) is present in the service, the calibration node can perform the necessary processing of the raw sensor data before pushing the data to the next node. An application that implements the positioning system can use the service to request the user to calibrate a specific sensor. Depending on the sensor, the user will have to perform a set of actions that store calibration data for this sensor. Other similar services exist for use cases such as user authentication.

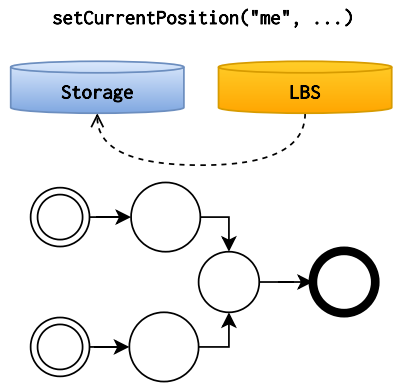
3.9 Location-based Service

To provide developers with an easy way to access the location data of users and objects, a location-based service (LBS) provides a simple application interface to retrieve the current position and a method to watch for changes in a position. In a stream processing network, it is more difficult to keep track of the changing location of objects due to the *process centric* approach rather than a *data centric* approach which is usually used in location-based services.

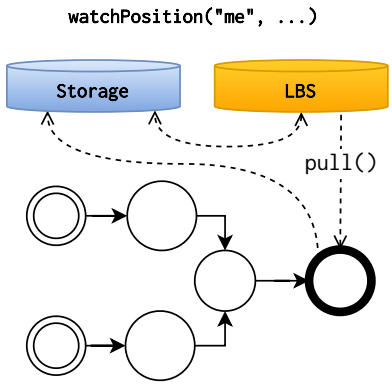
In OpenHPS, we create a service that works independently from the process network to offer developers a LBS interface that interacts with the process network behind the scenes. This service provides an interface similar to the Geolocation API [64]. Figure 3.11a illustrates the retrieval of the current position of an object through the persisted data service or by pulling the graph. When no up-to-date position is available for a particular object, the pull triggers an update of this information. Setting the current position of an object involves updating the persisted data service, as shown in Figure 3.11b. Watching for changes in the position of an object



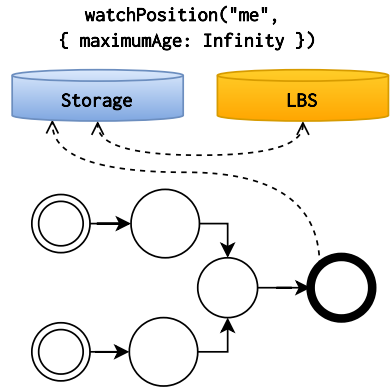
(a) Retrieve the position through the cache or by pulling the sink



(b) Set or update the current position of an object by modifying the cache



(c) Watch for changes in the position by monitoring the cache or by pulling the graph



(d) Watch for changes in the position by monitoring the cache when the maximum age is not important

Figure 3.11: Common LBS functionalities implemented in OpenHPS

can be done by monitoring the data service or pulling the graph when the data becomes outdated, as depicted in Figure 3.11c. In cases where the maximum age is irrelevant, monitoring the cache may be sufficient without requiring a pull, as shown in Figure 3.11d. These functionalities of the LBS service in a stream processing network provide developers with an easy way to interact with the network without manually pulling the graph to retrieve information.

3.10 Modularity

One of the non-functional requirements of the OpenHPS framework is its modularity. Due to the genericness of the framework, domain-specific functionalities

such as processing nodes for specific positioning techniques can easily cause a large and unmaintainable library. To solve this, we have split OpenHPS into a core component (`@openhps/core`) and various additional modules that provide more functionalities to the framework, such as additional data storage methods, algorithms or interfaces with hardware.

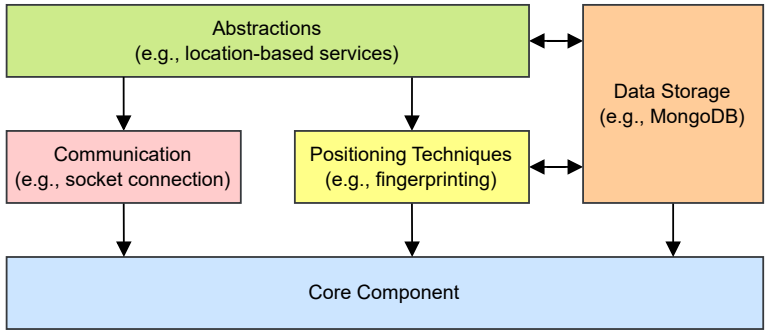


Figure 3.12: OpenHPS module stack

Figure 3.12 shows the module stack with the core component and additional modules linking to the core. Our four main module types are communication modules, which enable different methods to transmit information from one node of the graph to the other. This way, developers can create client-server applications that transmit information or create load-balancing graphs that distribute workload over multiple servers. The second type of modules is the positioning techniques and algorithms. Our core component contains various built-in features such as multilateration algorithms, but does not provide specific algorithms to create fingerprints or calculate a distance from RSSI. Next, we have the abstraction layers that mainly facilitate the use of the functionalities of the core module. This can be a location-based service that provides easy-to-use events or abstractions, such as the geospatial module that provides concepts for buildings, floors and rooms.

All the prominent modules that were created and released during our research are listed in Figure 3.13. In addition to the four module categories, we also created several miscellaneous modules to interface with other platforms such as Web APIs or hybrid mobile frameworks such as Cordova[5], React Native[6], NativeScript[7] and Capacitor[8]. Apart from the modules listed above, we also developed OpenHPS modules that were released as part of other projects. These modules include `@sembeacon/openhps`[9] and `@fidmark/openhps`[10].

[5] `https://cordova.apache.org`
[6] `https://reactnative.dev`
[7] `https://nativescript.org`
[8] `https://capacitorjs.com`
[9] `https://github.com/SemBeacon/openhps`
[10] `https://github.com/OpenHPS/FidMark/tree/main/openhps`

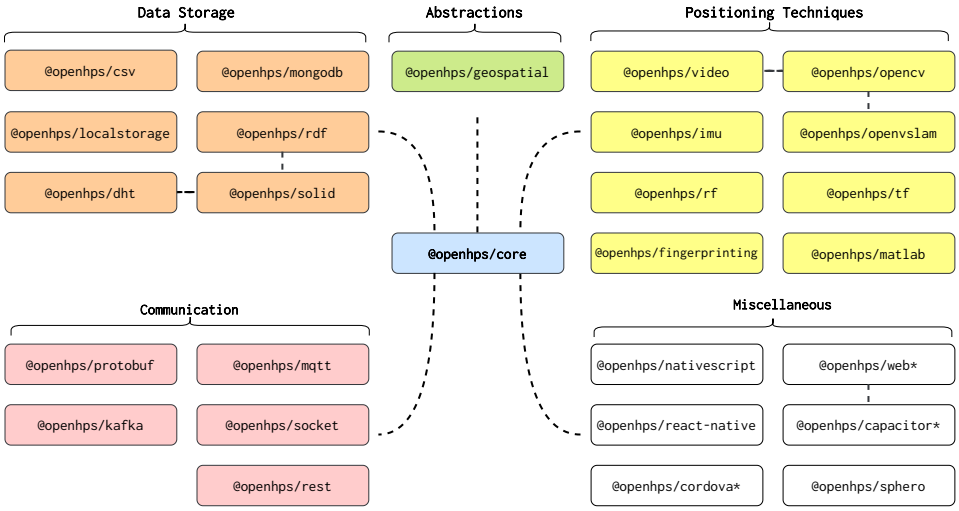


Figure 3.13: OpenHPS modules

Various implementations of positioning systems rely on server-side and client-side processing of sensor information. We developed a prototype where the server computes the position based on transmitted sensor data and data stored on the server. The client performs edge computing to enable more accurate position estimates while waiting for the server to compute the data.

OpenHPS as a framework itself is written in TypeScript, which requires a JavaScript engine. However, we created several modules, such as an MQTT module[11] or Apache Kafka module[12], which allow low-power hardware such as an ESP32 or Arduino to communicate with other processing nodes using the MQTT protocol. Using the serialisability of data, we can easily support communication between these low-power devices and more powerful servers or computation nodes. This allows for distributed processing and data exchange between different types of hardware.

3.10.1 Web-based Modules

Sensor data is one of the primary sources of information for positioning systems. In OpenHPS, we developed modules for interfacing with the sensor data of various platforms. However, with our aim for interoperability, creating new modules for every platform is not sustainable in the long run. The W3C community is slowly providing more and more access to sensor data via web applications. However, due

[11] `https://openhps.org/docs/mqtt/`
[12] `https://openhps.org/docs/kafka/`

to privacy concerns, this access is often limited and aims to mitigate the possibility for these sensors to be inadvertently used for tracking users[13].

Despite these limitations, our platform-specific modules focus on interfacing with Web APIs, providing additional contextual information in platforms capable of retrieving more information from sensors. Our `web-bluetooth` module uses the Web Bluetooth Scanning specification [183], this specification offers scanning for Bluetooth (Low Energy) devices. Discovered devices are uniquely identified using an internal identifier to ensure that these discovered devices cannot be identified across browsers or devices.

Figure 3.14: WebXR access to the computed depth data in OpenHPS

To access location data on the device, we provide a module which uses the Geolocation API. This is a Web API that provides access to the position, regardless of the technologies used to obtain this position [64]. The implementation of this API may return GPS location data if this is available in the platform, but may also provide location data based on the IP address. The module provides an active or passive source node that retrieves all the available information from the Geolocation API. More low-level sensor data can be obtained using `@openhps/web-sensors`, which enables developers to access the gyroscope, accelerometer and magnetometer.

[13] `https://www.w3.org/TR/generic-sensor/#location-tracking`

Modules such as `@openhps/webrtc` facilitate access to the camera feed and enriches the data with additional context needed to perform visual positioning. Finally, with `@openhps/webxr`, we facilitate access to raw data from the WebXR API. WebXR offers web developers the opportunity to develop AR or VR applications on the Web [171]. However, this data can also be utilised to develop positioning systems. Figure 3.14 demonstrates an example of OpenHPS where raw access to the depth data is provided. This information can be leveraged to perform visual dead reckoning.

All the Web modules are available in the monorepo[14]. Other modules, such as the CapacitorJS module, expand on these modules to enrich the data with additional context, such as the MAC addresses of scanned Bluetooth devices.

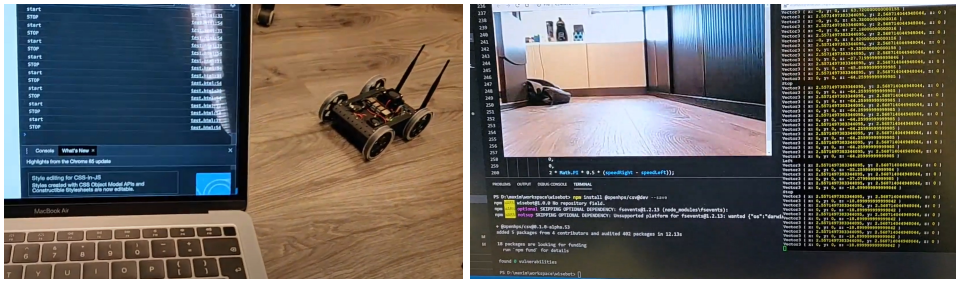


Figure 3.15: Robot running OpenHPS with a socket server that provides an endpoint for the source node and sink node for the camera output (integration test of the socket module in 2020)

By developing modules such as a REST server and client, and a socket connection for OpenHPS, we also facilitated communication between the browser and web server. Figure 3.15 illustrates an example of a robot with a source node that listens for socket connections. In the browser, users can easily connect to this server to broadcast data or movement instructions. This robot also has a sink node for the camera output and the estimation of its position based on dead reckoning.

3.11 Performance

While OpenHPS mainly focuses on the prototyping of positioning systems and the general abstraction of data and the systems themselves, we also focus on the performance of using the framework in real-world applications. Performance was evaluated in benchmark tests comparing the optimisations with regular processing methods of OpenHPS.

For the conceptual design of interoperable and discoverable indoor positioning systems, the performance of one system is less crucial. However, to validate that

[14] `https://github.com/OpenHPS/openhps-web`

our processing pipeline represents a real-world scenario, as well as to validate that the use of OpenHPS is viable for creating proof of concepts, we still consider the performance of the framework to a certain extent. We developed OpenHPS in TypeScript to enable cross-platform deployment of the system. This cross-platform compatibility also comes at the cost of decreased performance, which is a balance we have to find for the sake of portability.

3.11.1 Distributed Processing

Using our communication modules, such as MQTT, we can easily support distributed processing between different hardware nodes. By offloading computation to multiple nodes, we can improve the overall performance of the system. This is especially useful in scenarios where real-time processing is required and a single node may not have enough resources to handle the load.

By default, as mentioned in Section 3.6, OpenHPS uses JSON to serialise transmitted data. Modules such as `@openhps/protobuf`[15] offer optimisations using *Protocol Buffers* [184] to reduce the size of serialised data. This module analyses all serialisable objects and data to produce a custom protocol that addresses objects and fields using as few bytes as possible. This ensures that data can be transmitted efficiently, but it requires both nodes to have prior knowledge of the generated protocol. More details on the module are given in Appendix A.3.3.

3.11.2 Parallelism and Workers

Individual nodes or parts of the graph can be configured to run in parallel. Depending on the deployment, we use Web Workers [185] in the browser, and worker threads with Node.js. The workers process data in the graph in a separate JavaScript instance. Data frames, represented as JavaScript (JS) objects, are first serialised to JSON before sending them to a worker. In the worker, they are deserialised to a JavaScript object and processed by the processing nodes or graph that the worker is responsible for. After the worker is finished, the output data frame is serialised back to JSON and sent to the main thread, where it is deserialised before being pushed to the next node in the graph.

Figure 3.16 illustrates the internal workings of a `WorkerNode` with a pool of workers. A `WorkerNode` is a node within a graph on the main thread that spawns and manages a pool of workers to parallelise one or more chained processing nodes. Each worker in the pool performs the same task (i.e., one defined processing node or graph). The processing of a data frame is performed by the first available worker in the pool, and is only performed by that one worker. As depicted in Figure 3.16, a data frame is pushed as a JS object to a `WorkerNode`. The node first serialises the data to JSON in the main thread before sending this JSON data to an available

[15] `https://github.com/OpenHPS/openhps-protobuf`

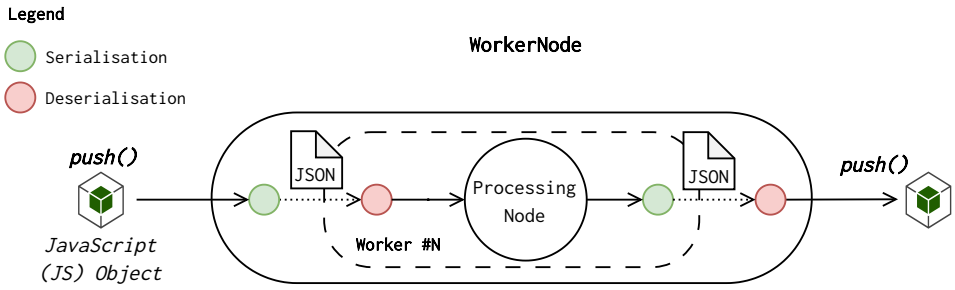


Figure 3.16: `WorkerNode` serialisation and deserialisation steps

worker. A worker has to deserialise the data (indicated as the red circle) before processing it, and serialising and transmitting it back to the main thread. In the main thread, the node will deserialise the data before pushing the JS object to the next node in the graph. Workers have access to the services on the main thread for nodes that require synchronisation. However, this feature was not included in the benchmark detailed below.

We acknowledge that these two stages, in which data is serialised and deserialised twice, result in overhead when processing data over multiple workers. Our decision to rely on this serialisation is to ensure that nodes developed in the OpenHPS framework are compatible throughout the framework, regardless of whether they are used within a worker. To satisfy this, we must ensure that the data that is being processed does not lose any context (i.e., types) when transmitted to a worker.

As a simple demonstration of our worker node, we created a processing node that generates 5000 prime numbers for every received frame. This task was chosen for its processing time and is not relevant for a positioning system. Each run of the task will generate the same set of numbers, and the task is therefore not distributed among nodes, and no synchronisation is needed between workers. This test was conducted in the first alpha version of OpenHPS in 2020 using an Intel i7-6700HQ laptop CPU with 8 logical cores, running Node.js 14.10. These 5000 prime numbers can be generated 237.03 times per second without the overhead of data frames, objects and services. The data frames that we push contain a source object, position and velocity to simulate the amount of data normally serialised and communicated between the main process and workers. However, the contained data does not affect the time it takes to execute the task.

Each data frame was populated with dummy data objects, positions and orientations to simulate real-world conditions. The benchmark aims to showcase that the processing of multiple consecutive frames can be parallelised over multiple nodes. Such a scenario occurs when processing data from multiple users who need to be tracked independently from each other, which is one of OpenHPS' strengths compared to frameworks such as ROS [61] that only track one robot/object. Since

the data has to be serialised and deserialised on the main thread, this becomes the bottleneck in the processing of frames, as indicated by the non-linear speed-up. Table 3.1 shows the results of our benchmark with one worker assigned to each logical CPU core. Performance is measured in frames per second (FPS) represented by the amount of processed data frames received by the sink of our model. For each individual worker, we indicate the speed-up compared to the sequential implementation. The overhead shown with a single worker is due to the serialisation and deserialisation of data, an operation that is not required when pushing in a sequential network.

A warm-up initiated each test to prepare the spawned web workers (i.e, nodes). For each test, 100 frames were created and pushed through the *worker node*. Tests were conducted 100 times (each with a warm-up) to determine the average frames per second. The standard deviation is the error (in frames per second) of those 100 runs.

#workers (#w)	FPS	Deviation	Speed-up
Sequential	229.04	1.19%	-
1	200.74	0.67%	0.88
2	389.44	0.56%	1.70
3	512.42	0.92%	2.24
4	616.29	1.15%	2.69
5	671.00	0.59%	2.93
6	746.07	0.67%	3.26
7	801.32	0.90%	3.50
8	822.47	0.69%	3.59

Table 3.1: `WorkerNode` benchmark

Newer versions of OpenHPS offer serialisation optimisations which may affect the performance benchmarks. In Table 3.1, we can observe a clear trend of increasing performance as more worker nodes are utilised. The speed-up values indicate the improvement in computation speed compared to the sequential implementation. It is important to note that the overhead introduced by serialisation and deserialisation is also reflected in the results, as seen in the drop in FPS when moving from sequential processing to using a single worker.

In 2025, we performed the benchmark again, this time with a second task that has increased complexity. Our aim with this new benchmark was to showcase that the serialisation and deserialisation in the main thread is the bottleneck that prevents linear speed-up. The tests were performed on an AMD Ryzen 9 5900X with 12 cores. Similar to the previous test, 100 frames were pushed, with each run being executed 100 times. Our speed-up results are shown in Figure 3.17 with our data being showcased in Tables 3.2 and 3.3. From our results, we can observe that

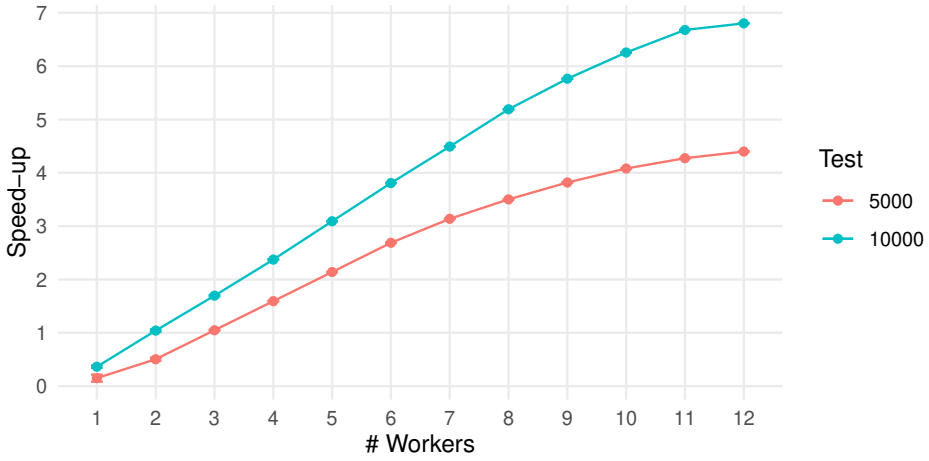


Figure 3.17: Speed-up of two different tests (5000 and 10 000 prime numbers)

as the task becomes more complex (generating 10 000 prime numbers for each frame instead of 5000), the speed-up becomes more linear with the serialisation and deserialisation in the main thread having a more minimal impact on the total duration. From the results in Table 3.3, it can be observed that the speed-up does not increase linearly with the number of workers. This is most likely due to the remaining bottleneck of the `WorkerNode` that still has to serialise and deserialise the data in the main thread, regardless of the throughput of the workers.

#w	FPS	Deviation	Speed-up
Seq.	654.33	0.19%	-
1	98.05	6.53%	0.15
2	330.19	2.68%	0.50
3	685.68	1.06%	1.05
4	1042.68	0.42%	1.59
5	1400.12	0.34%	2.14
6	1758.73	0.09%	2.69
7	2052.77	0.11%	3.14
8	2291.66	0.10%	3.50
9	2498.52	0.11%	3.82
10	2670.44	0.14%	4.08
11	2796.87	0.15%	4.27
12	2878.37	0.17%	4.40

Table 3.2: Benchmark for 5000 prime numbers per frame

#w	FPS	Deviation	Speed-up
Seq.	223.52	0.48%	-
1	81.44	2.18%	0.36
2	232.76	2.25%	1.04
3	379.16	1.30%	1.70
4	530.38	0.72%	2.37
5	691.02	0.39%	3.09
6	851.18	0.20%	3.81
7	1004.25	0.17%	4.49
8	1160.47	0.14%	5.19
9	1288.46	0.18%	5.76
10	1398.37	0.18%	6.26
11	1492.94	0.17%	6.68
12	1520.57	0.36%	6.80

Table 3.3: Benchmark for 10 000 prime numbers per frame

3.11.3 Native Library Bindings

Algorithms that require a lot of computational power, such as computer vision algorithms, already have a lot of libraries available that focus on performance. In OpenHPS, we provide native bindings to these libraries, allowing developers to easily integrate them into their positioning systems without sacrificing performance. In the case of our `@openhps/opencv` module, which adds bindings to the OpenCV library [186], we only include a memory pointer to images within the data frames that are pushed through our process network.

3.12 Discussion

In this chapter, we presented OpenHPS, an open-source framework for developing hybrid positioning systems. We started by defining the framework requirements in Section 3.2, where we also introduced our system actors: a tracked actor, tracking actor, calibration actor and computing actor. Two of our main requirements were for positioning systems developed with OpenHPS to run in de decentralised context and for OpenHPS not to be designed for one specific platform.

Using the requirements and actors, we created a data structure that can be applied to various positioning systems. This data structure transfers data through a processing network represented as a graph. Our implementation of this data structure and graph representation was made in a TypeScript framework. Due to the serialisability of the data structure, we can ensure that parts of the processing network can run distributed over multiple workers, both local as well as remote. This ensures that the framework is scalable and can run on different types of hardware setups. TypeScript also enables us to develop positioning systems that work on the Web, on smartphones, on servers or even on embedded devices.

We conducted a benchmark to evaluate the performance of our framework with different numbers of worker nodes assigned to each logical CPU core. The results showed a clear trend of increasing performance as more worker nodes were utilised. The speed-up values indicated the improvement in computation speed compared to the sequential implementation. However, we also observed overhead introduced by serialisation and deserialisation of data, which impacted the performance, especially when moving from sequential processing to using a single worker.

Note that in Chapter 6, we provide several examples and use cases where OpenHPS is used in combination with other contributions discussed in the dissertation. Furthermore, we utilised the results of the data structure defined with OpenHPS to further define a generic data structure of positioning systems in an ontology, which is detailed in Chapter 4.

Data is a precious thing and will last longer than the systems themselves.

– Tim Berners-Lee

Chapter 4

Interoperable Positioning Systems

Interoperability of data is defined as data that can be accessed, read and above all understood by data processors other than the one that generated the data [107, 17, 18]. It is a concept introduced in the early days of computing [187]. Back in the early 1960s, interoperability was more broadly defined as a means to enable computers to work together, which was an issue with the first network-connected databases that each required their own protocols to extract or insert data. Over the years, interoperability has evolved to encompass various levels and types of data exchange between systems, including *syntactic*, *semantic* and *process* interoperability [30, 108, 109, 110]. Syntactic interoperability ensures that data can be exchanged and interpreted correctly between systems, while semantic interoperability focuses on ensuring that the exchanged data is understood in the same way by different systems without ambiguity. Process interoperability goes a step further by ensuring that the processes involved in handling the data are also compatible and both syntactically and semantically interoperable. These different levels of interoperability are essential for the seamless exchange of positioning data.

Forcing interoperability on applications, including services and systems such as indoor positioning systems, helps to preserve the data these systems produce. The quote from the founder of the Web, Tim Berners-Lee, at the beginning of this chapter fits well with this goal, since the data these systems produce is more valuable than the systems that produced them. With artificial intelligence and machine learning becoming more prevalent in the field of indoor positioning systems, the need to collect and understand data needed for training the next generation of indoor positioning systems becomes the next challenge.

Definition 5: Interoperable Positioning System

We define an interoperable positioning system as a hybrid or integrated positioning system that can consume and produce data in a way that enables other systems and services to access, read and understand this same data without ambiguity.

In the context of positioning, we define interoperable positioning systems in Definition 5 as systems that produce data (e.g., a position or orientation) that can be accessed by other positioning systems and can be read and understood by those systems. In addition to creating interoperable data, these systems should also be able to describe themselves to provide more context on how the data that they provide is computed or processed. Knowing the system that generated the data enables semantic reasoning about the quality or usefulness of the data.

Designing interoperability for positioning systems allows any positioning system to make use of data from other technologies. This entails that interoperable positioning systems should also be considered hybrid systems, as the underlying technologies may come from one or more data sources and may combine different algorithms.

4.1 Methodology

Our methodology for designing interoperable positioning systems is based on the requirements defined in Chapter 3. In these requirements, we specified that a generic, or more specifically, hybrid positioning system should run on various platforms and support a wide range of algorithms. Using these requirements and our preliminary research detailed in Section 2.4.3, we have chosen to design our interoperable positioning systems using the Semantic Web. The Semantic Web, or more specifically linked data, offers a way to represent and exchange data in a standardised and interoperable manner. By using ontologies and vocabularies to define the meaning of data elements, we can ensure that data produced by a positioning system can be easily understood and utilised by other systems. Furthermore, by leveraging existing semantic technologies such as RDF and SPARQL, we can enable seamless integration and querying of data from different sources without implementing new proprietary protocols.

To enable this, we have split our design for interoperable positioning systems into three contributions: (contribution 1) a generic positioning system ontology for describing different positioning technologies and data, (contribution 2) a serialisation framework for mapping our generic OpenHPS framework to RDF data, and (contribution 3) a solution for enabling transparent data exchange using personal data vaults.

Our design of a generic positioning system ontology is similar to the Linked Open Terms (LOT) methodology [188]. However, the LOT methodology design for ontologies was only published after we started with our ontology design. The terminologies and generic concepts we defined are based on existing specifications and were already investigated in research question RQ1, previously discussed in Chapter 3. Our solution for decentralised data exchange (contribution 3) was chosen based on our preliminary investigation. In this investigation, we explored various solutions such as OpenLS [112] and SemanticLBS [113], which offer decentralised location-based services. However, these solutions were service-centric and did not fully meet our requirements for transparent data exchange between interoperable positioning systems. Thus, we propose using decentralised personal data vaults as a solution to enable user-centric interoperable positioning systems. Our serialisation framework in (contribution 2) is meant to provide a mapping of the already defined input and output data of hybrid positioning systems to our generic positioning system ontology (contribution 1). This mapping also offers validation of the completeness of our ontology.

We also validated our ontology through ontology alignments and extensions. Alignments enabled us to verify whether our ontology can be integrated with existing and commonly used ontologies available on the Semantic Web, while the extensions allowed us to verify that our generic positioning system ontology was generic enough to enable extensions. Our solution for user-centric, interoperable positioning systems was validated through a proof-of-concept application that incorporates three different types of implementations. This proof of concept is further detailed in Section 6.4.

4.2 Positioning System Ontology

With research question RQ2, our goal is to design interoperable positioning systems. In Definition 2, we defined an interoperable positioning system as a hybrid or integrated positioning system that enables other systems and services to access, read and understand this same data without ambiguity. To achieve this goal and to aid with the interoperability of positioning data and the processes involved in computing this data, we designed the Positioning System Ontology (POSO) along with several extensions that add additional semantic terminology to describe different types of positioning systems and techniques.

In Figure 4.1, we provide a general overview of a positioning system and related components. A positioning system is deployed at a particular location or area that is meant to be covered. This can be a building, an area outdoors or even a location on a game board that does not have to be mapped to any geographical boundaries. Each positioning system uses a set of algorithms and technologies to help compute a position. Finally, with positioning systems modelled based on POSO, we aim to

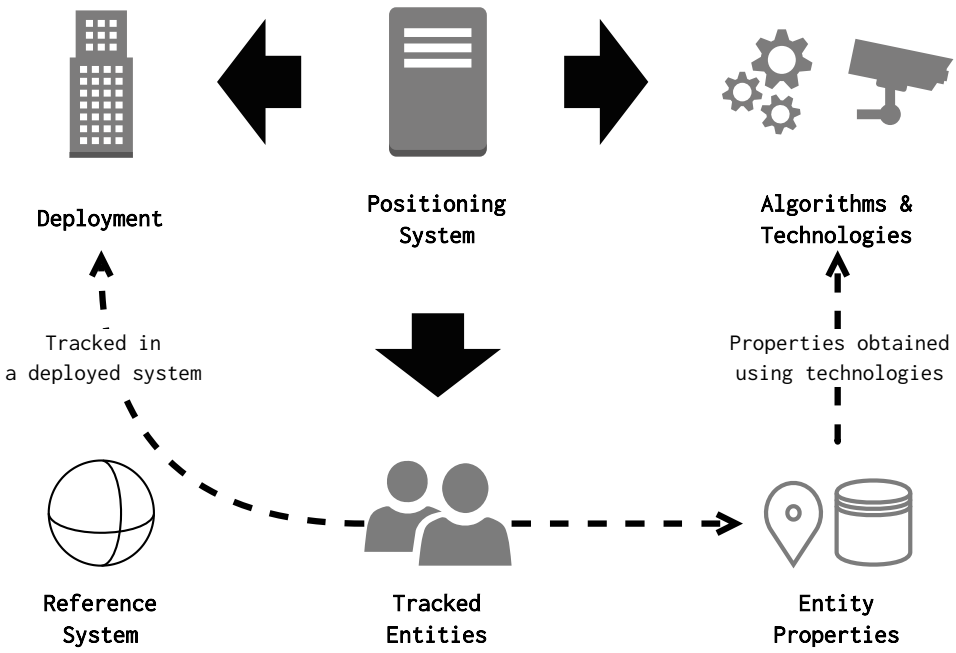


Figure 4.1: Basic structure of a positioning system that tracks entities

track the position, orientation and other properties of one or more entities. These properties can be anything that is of relevance to the system and are obtained using the techniques implemented by the positioning system. Spatial properties of a tracked entity are located within the deployment using an optionally defined reference system.

4.2.1 Ontology Design

We first searched for existing ontologies that could potentially serve as a basis for describing positioning systems. We used academic research as well as the Linked Open Vocabularies (LOV) [189] in combination with common terminologies.

Based on our investigation of existing ontologies, we designed POSO with the Semantic Sensor Network (SSN) as a top-level ontology [130] together with the Sensor, Observation, Sample and Actuator (SOSA) ontology [131]. Combined, SOSA and SSN provide an ontology for linking sensors, actuators, observations, samplers (i.e., a device to transform a sample) and the systems needed to process this sensor data into an output.

Our design decision to use SOSA and SSN was based on their generality to represent *systems* and the flow of information within such systems. In general, positioning services are often part of a larger system as a whole. SSN offers the ability to describe a positioning system and utilise this description as a subsystem

(`ssn:hasSubSystem`). Likewise, hybrid or integrated positioning systems that perform fusion of multiple smaller (positioning) systems can utilise the design of SSN to identify the subsystems that are being integrated. Furthermore, the popularity and widespread use of SSN and SOSA makes it a suitable choice for achieving interoperability among different positioning systems. Finally, the ontology is co-edited by the Open Geospatial Consortium (OGC), making it a potential candidate for future alignment with existing geospatial standardisations.

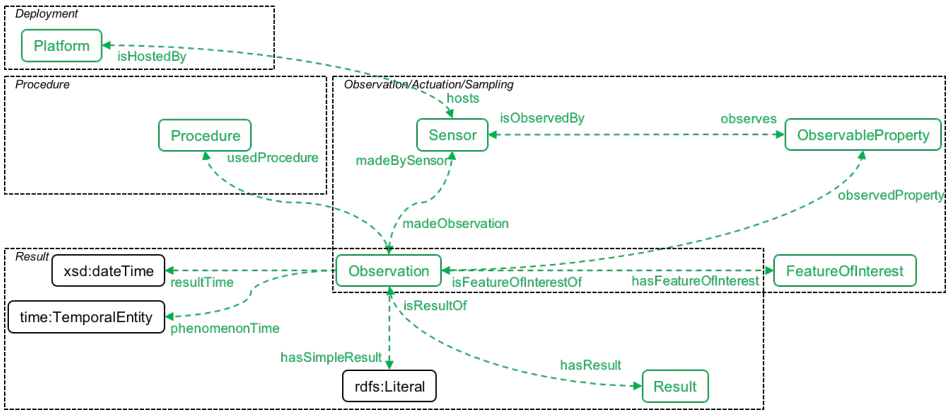


Figure 4.2: Main SOSA classes [190]

Figure 4.2 illustrates the main classes of the SOSA ontology from an *observation* perspective. Observations of information are made by sensors that observe certain properties. These observations may be obtained using a set of procedures which are part of a system.

To validate that SOSA was extensible enough to accommodate the description of a positioning system, we first designed a proof of concept using SOSA in Section 6.4. In this proof of concept, we demonstrated how we could leverage SOSA for describing a positioning system.

SOSA and SSN provide a stable core ontology that could enable the modelling of a positioning system with its deployment, the used sensors, procedures, entities and as well as the observable properties of those entities. However, as these ontologies are meant to be used as core ontologies, they do not offer any semantics for expressing the accuracy of individual observations, the different types of algorithms that are relevant for positioning or how the results should be represented to be interoperable.

Our POSO ontology has been designed with the common data requirements of various positioning system technologies [54, 2], datasets [191, 192, 193] and frameworks [33, 160, 61] in mind to cover all types of systems without over-complicating the modelling of the data. In Sections 6.4, 6.6, 6.7 we provide several proof of concept implementations that demonstrate the use of POSO in various scenarios.

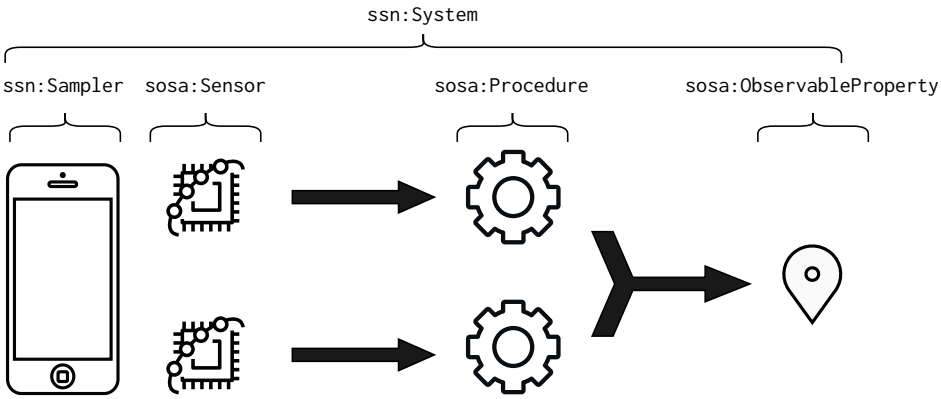


Figure 4.3: Example mapping of a positioning system to SSN and SOSA

Figure 4.3 illustrates how we map a generic positioning system to the SSN and SOSA ontologies. In the illustration, a system is a broad concept that can describe anything from a positioning system to a thermostat. To ensure that a positioning system is more specifically described, the POSO ontology acts as an intermediary layer that connects the specifics of a positioning system to the more generalised concepts provided by SSN and SOSA. The complete system is represented by all the samplers, sensors and procedures that output computed data. This computed data is defined as *observable properties* and is detailed in Section 4.2.2. The procedures depicted as `sosa:Procedure`(s) in Figure 4.3 will be discussed in Section 4.2.4.

4.2.2 Observable Properties

With our POSO ontology, we aim to support concepts for defining a generic position, orientation, velocity, acceleration and the sampling of this data. We extended the `sosa:ObservableProperty` to express different types of position, orientation, velocity and acceleration. For expressing observation-based sensor data, we use the SOSA ontology together with the QUDT ontology for expressing Units of Measure, Quantity Kinds, Dimensions and Data Types [194]. Each observable property defined in POSO can also be used as a result within a SOSA observation, with a set of predicates that express the result. This enables expressing a fixed position of a feature of interest as shown later in Listing 4.6. The proposed vocabulary should support the following three main goals:

- **Sensor fusion**: High- and low-level sensor fusion should be possible based on the data [102]. High-level fusion, also called decision-level fusion, consists of merging processed data from multiple sources, while low-level sensor fusion is the use of multiple sources of raw sensor data. Both fusion levels require additional knowledge of how the data has been obtained and its quality. In the context of high-level fusion in a positioning system, the additional

semantics include the accuracy as well as the techniques used to obtain the data. Using this knowledge, other systems can prioritise the observations to be used.

- **Historical data**: Positioning systems make use of previous information to predict future movement [195]. These predictions can be used to improve the calculation of the next position. To support this technique, historical positioning data should be available.

- **Granularity**: The position of an entity should be offered with varying ranges of granularity without causing conflicts with the decision-level sensor fusion. This enables use cases where observations of a minimum or maximum accuracy can be separated in a different triple store, further enabling access control to these individual stores.

In addition, we want observable properties to be extensible in the future. This enables other vocabularies to extend these properties with additional details specific to their domain, while still maintaining compatibility with POSO. By defining a set of common observable properties that cover a wide range of positioning data, we ensure that different systems can easily integrate and communicate with each other. In the following sections, we will delve into the implementation of POSO and the different types of observable property classes we provide.

Absolute and Relative Positioning

When working with absolute positions in a geographical coordinate system, we make use of GeoSPARQL's geographical position representation by the Open Geospatial Consortium (OGC), which supports GeoJSON, Well-known Text Representation (WKT), Geography Markup Language (GML) and other representations [117, 70]. However, for absolute positions that should not be expressed as geometric coordinates, we use the QUDT ontology [194] to express Cartesian coordinates. POSO provides the concepts of `:xAxisValue`, `:yAxisValue` and `:zAxisValue` to express a `qudt:QuantityValue` in two or three dimensions.

Despite using simple Cartesian coordinates for a non-geographic position, a reference frame is still required to indicate how the Cartesian coordinates relate to each other. Similar to a reference frame in a geographical context, the reference frame allows the 2D or 3D position to be converted to other reference spaces such as a geographical context while still enabling the use of a positioning system that is only meant to operate in a specific context (i.e., an engineering reference frame as defined in ISO 19111 [196]). Defining a reference system is already well covered in GeoSPARQL [167]. To define the reference system of a `sosa:Result`, the `:hasSRS` or `:hasCRS` properties can be used.

For expressing a location that is covering a less specific, larger 2D or 3D area, we still request the use of an absolute position, but provide the ability to indicate the accuracy as either a one-dimensional (i.e., distance) or polygonal coverage. Accuracy may be used by processing algorithms to determine the reliability of the data and make informed decisions based on it.

Orientation

As detailed in Section 2.1, orientation is an important aspect of a positioning system. It not only offers the final state of direction after a rotation of an object or person, but is also required by many positioning algorithms to determine a position.

As we aim to create a generic ontology, we have chosen to support any concept that can identify the orientation around three axes. POSO provides three extensions of the `:Orientation` class, including an `EulerOrientation` class, `AxisAngleOrientation` and `QuaternionOrientation`. For Euler orientations, we provide the concept of pitch, roll and yaw as well as the order in which the Euler orientations should be applied. In addition to Tait-Bryan Euler angles, we provide concepts to describe proper Euler angles [197].

Velocity and Acceleration

Active positioning systems make use of an object's velocity to determine a position and orientation based on its momentum. This procedure, called dead reckoning, uses an entity's last known location together with its angular and linear velocity to determine the new position and orientation at a later timestamp. POSO adds the concept of `:Velocity` with `:LinearVelocity` and `:AngularVelocity` as subclasses, as well as the momentary acceleration that is often returned by a common inertial measurement unit (IMU).

4.2.3 Observations and Accuracy

Individual observations and different levels of granularity can be expressed for all properties. SSN-Systems [198], an extension of the SSN ontology, supports the description of a system's properties, capabilities and conditions. While this enables the semantic description of the potential properties (i.e., accuracy, precision and operating environment) of a positioning system, it does not provide information on the individual observations. For a positioning system, the spatial accuracy can vary depending on the implemented procedure, the amount of sensor data, as well as the accuracy of that data.

The accuracy of any observation can be expressed via `:hasAccuracy`, a subproperty of `ssns:qualityOfObservation`[1] that can be applied to an observa-

[1] `ssns:` is the prefix for SSN-Systems [198]

tion or individual result. Alternatively, for expressing the accuracy of spatial data (i.e., absolute or relative position), the `ogc:hasSpatialAccuracy` from the GeoSPARQL 1.1 draft [128] can be used to express a QUDT quantity value. Further, to express the aimed accuracy of an observable property, the `ssns:Accuracy` class can be used to indicate that the accuracy applies to the position.

Creating an observation for every calculated position provides context on historical data that can be used. The semantics of trajectories, such as segmentation, map matching and additional post-processing context [199] lie beyond the scope of our positioning system ontology. However, as each observation is a momentary timestamped result, they indirectly support the modelling of a trajectory space and time path [200]. Despite being able to infer trajectories from a list of observations, the overhead of describing these individual observations is significant. Future work should analyse how we can better represent trajectories in RDF.

4.2.4 Positioning Algorithms and Techniques

The SOSA ontology describes a `sosa:Procedure` as a workflow, protocol, plan, algorithm or computational method to make an observation, sample or change the state of the world[2]. In a positioning system, we identify a procedure as a workflow that processes sensor data to an intermediate result or observation.

A positioning system can use a broad range of techniques to calculate a position. While it might perform generic processing on raw sensor data, semantically describing the main techniques that are involved in the processing improves the reasoning that can be performed on the sampled data as well as its priority for decision-level sensor fusion. To illustrate this, we provide the example of an indoor positioning system (IPS) that uses simple QR codes for room check-ins and an IPS at the same location site that uses Bluetooth beacons. Without knowledge of the techniques used to determine a position, the accuracy of the position at a given time cannot be determined reliably. While the Bluetooth positioning provides a continuous output with varying accuracy, the QR scanning only provides a very high accuracy position when it is scanned, as the person will be near the code to scan it.

Figure 4.4 illustrates the main classes in the POSO v1.1 ontology, consisting of positioning techniques, observable properties and the different systems. Our initial implementation can be found in Appendix B.1. In POSO, we subdivide a procedure over multiple different main categories that are based on the work of Liu et. al [54] and Gu et. al [2]:

- **Cell identification**: This covers all techniques that detect the position of an object when it is close to an object with a known position. Existing solutions

[2] `https://www.w3.org/TR/vocab-ssn/#SOSAProcedure`

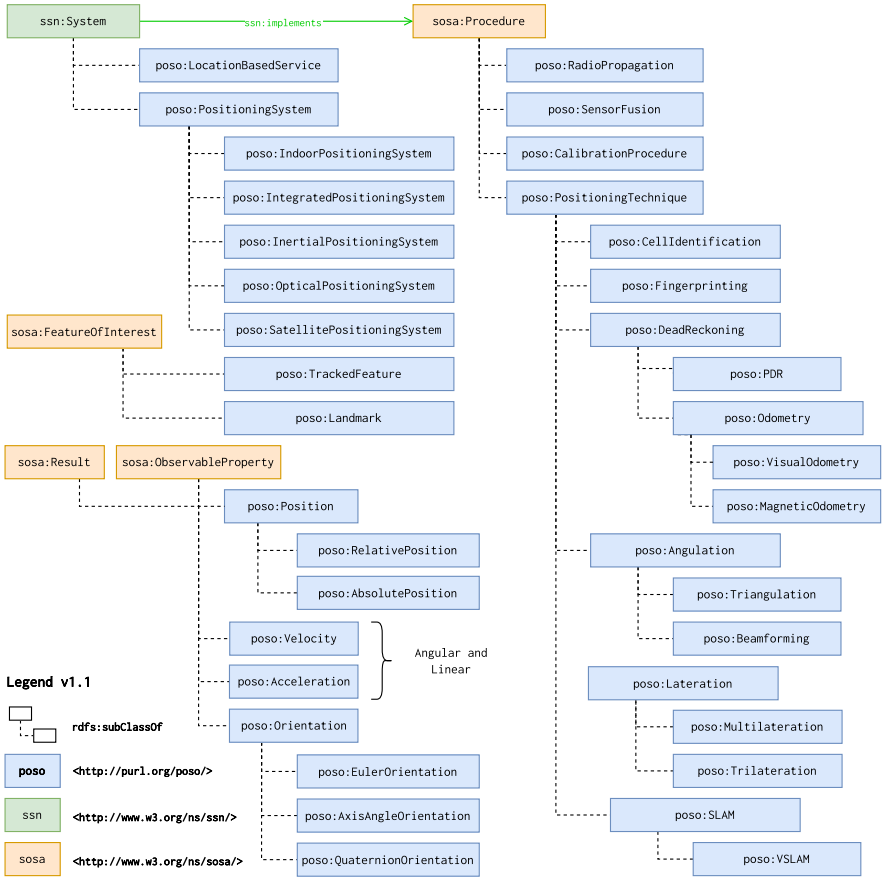


Figure 4.4: Main POSO ontology classes

range from radio frequency proximity to implicit positioning, such as the act of scanning a QR code at a known fixed location.

- **Dead reckoning**: The movement of an object can be used to determine its drift in space. This technique, called dead reckoning, can be a positioning system on its own, identified as an inertial positioning system [46], but can also form part of another technology such as Assisted GPS [48]. A subclass of dead reckoning is **odometry**, which is a set of techniques that use sensor data to detect the change in position. This can be sensor data from motion sensors, visual observations or other environmental data such as magnetic interference [201].

- **Fingerprinting**: Scene analysis techniques such as fingerprinting, where sensor data is matched to a grid of positions and can be used during the

setup of the positioning system. Each scene analysis at a position is called a *fingerprint* and is used during the online tracking stage to determine a position. The sensor data will be matched to the fingerprint that most closely resembles this data. POSO expresses a fingerprint as a subclass of `sosa:FeatureOfInterest` under the term `:Fingerprint` that requires to have a position to qualify as a fingerprint. This allows positioning systems that make use of this scene analysis to semantically describe the system's setup.

- **Simultaneous localisation and mapping**: In simultaneous localisation and mapping (SLAM), a sensor determines features that are tracked during movement. By tracking these features, it can determine the drift while simultaneously using the features to construct a map of the environment [202]. SLAM can be subdivided into Visual SLAM [50] where image sensors are used to track features as opposed to LiDAR sensors.

- **Angulation**: Angulation covers all angle-based positioning techniques such as **triangulation**, which describes positioning techniques that use angles to determine a position between two or more landmarks with a known position.

- **Lateration**: Lateration covers all distance-based positioning techniques such as **multilateration**, which involves determining the position of an object by measuring or approximating the distance from the object to multiple fixed points with known positions and **trilateration**. Other than multilateration, trilateration specifically uses three (for 2D) or four (for 3D) relative distances.

- **Sensor fusion**: To specify how multiple positioning systems or sensors are used together, a sensor fusion procedure category defines procedures where observations from multiple different (sub)systems are merged. This fusion technique can further make use of additional available context.

A visual example usage of the POSO ontology is given in Figure 4.5 where a tracked feature (`:me`) has a velocity, orientation and position as observable properties. A single observation of a position has a specific result and is made by a certain positioning system, in this case `:OfficePositioning`, which is a type of indoor positioning system with a geographical coordinate reference system.

In the FidMark ontology [41], we extended the available algorithms and categories to include pose estimation algorithms which are commonly used in computer vision and robotics. As illustrated in Figure 4.6, we classify these pose estimation algorithms as sub-classes of positioning techniques, while other algorithms, such as marker detection, are sub-classes of generic *procedures*. Appendix B.2.3 provides more information on other classes and properties in FidMark that are aligned with the POSO ontology. Additionally, Appendix A.3.1 provides an example of the FidMark ontology in the OpenHPS framework.

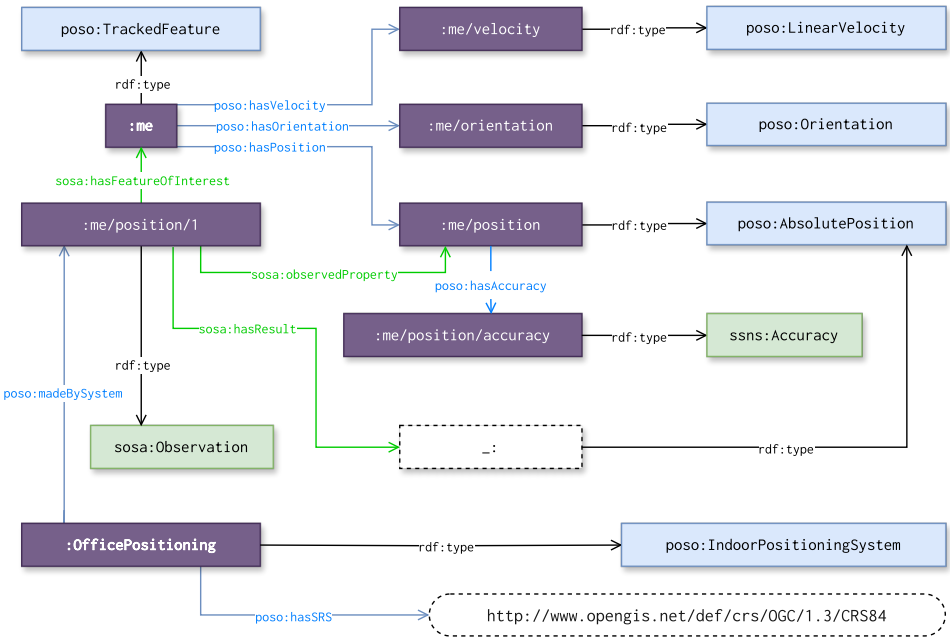


Figure 4.5: POSO example usage

4.2.5 Common Algorithms and Systems

The `poso-common` alignment module provides individual common positioning algorithms, systems and data used in positioning systems categorised under the classes defined in POSO. It describes seven satellite positioning systems [47]; known platforms such as IndoorAtlas[3], AnyPlace [160], Robot Operating System (ROS) [61] and our own OpenHPS framework along with individual algorithms for common positioning techniques. With the provided `poso-common` alignment module, we want to offer a foundation of algorithms and techniques that can easily

[3] `https://www.indooratlas.com`

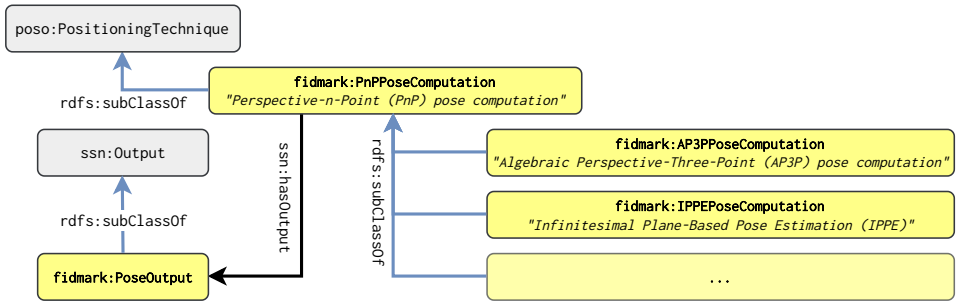


Figure 4.6: Pose computation procedures in the FidMark ontology

be used to describe complete positioning systems. Future work should focus on expanding these algorithms, along with more detailed descriptions of their input and output shapes. In a hybrid or integrated positioning system, the use of these common algorithms can provide insights into what observations to use in the fusion process.

4.2.6 Demonstration

To demonstrate the use of POSO to semantically model multiple positioning systems, we provide an example of a campus positioning system for the indoor as well as outdoor tracking of students. Our fictional setup consists of three individual systems: an outdoor positioning system using GPS, an indoor positioning system using Wi-Fi fingerprinting and a hybrid position system that makes use of the indoor and outdoor tracking subsystems by using a high-level sensor fusion technique.

```
1  @prefix poso: <http://purl.org/poso/> .
2  @prefix poso-common: <http://purl.org/poso/common/> .
3  @prefix ssn: <http://www.w3.org/ns/ssn/> .
4  @prefix sosa: <http://www.w3.org/ns/sosa/> .
5  @prefix dbr: <http://dbpedia.org/resource/> .
6  @prefix ogc: <http://www.opengis.net/ont/geosparql#> .
7  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
8  @prefix qudt: <http://qudt.org/schema/qudt/> .
9  @prefix unit: <http://qudt.org/vocab/unit/> .
10 @prefix ssns: <http://www.w3.org/ns/ssn/systems/> .
11 @prefix schema: <http://schema.org/> .
```

Listing 4.1: Prefixes used in the demonstration examples

We start by semantically describing the technical setup of the fictional deployment of the three positioning systems on our campus. Additional domain-specific ontologies such as IndoorGML [149] can be used to describe the physical context of these deployments. Throughout our examples, we make use of the prefixes defined in Listing 4.1.

In Listing 4.2 we create an outdoor campus positioning system that uses GPS. Indoors, we deploy a system that uses k-NN fingerprinting for Wi-Fi access points. For the integrated positioning system on lines 14 to 17 that uses both the outdoor and indoor system, we add the two individual systems as subsystems with an additional procedure on how the high-level fusion of these two systems is performed.

The entity that is being tracked by the campus positioning system is configured in Listing 4.3. Each feature of interest, which we identify as our *tracked feature*, has multiple observable properties. A property predicate such as the `:hasPosition` on line 3 can be used multiple times to represent a position with different levels of

```
dbr:Some_Unversity a ssn:Deployment .
<deployment/building_a> a poso:IndoorDeployment, ogc:Feature ;
  rdfs:label "Building A"@en ;
  ogc:hasGeometry [ a ogc:Geometry ;
    ogc:asWKT "..."^^ogc:wktLiteral ] .
<system/OPS> a poso:LocationBasedService ;
  rdfs:label "Outdoor campus positioning"@en ;
  ssn:hasSubSystem poso-common:GPS ;
  ssn:hasDeployment dbr:Some_University .
<system/IPS> a poso:IndoorPositioningSystem ;
  rdfs:label "Indoor campus positioning"@en ;
  ssn:hasDeployment <deployment/building_a> ;
  ssn:implements poso-common:KNNFingerprinting .
<system/CampusPositioning> a poso:IntegratedPositioningSystem ;
  rdfs:label "Hybrid campus positioning system"@en ;
  ssn:hasSubSystem  <system/OPS>, <system/IPS> ;
  ssn:implements poso-common:WeightedAccuracyFusion .
```

Listing 4.2: Positioning system setup

granularity. In linked data front ends with data access control, such as Solid [139], these levels of granularity can control who is able to access a property with a certain accuracy. By specifying the accuracy of these properties along with possible other semantic information, the information can be used in queries to determine which property offers the required accuracy.

Further, in Listing 4.4 we show an observation created by the outdoor positioning system. The GPS provides a latitude and longitude that we output using the OGC GeoSPARQL 1.1 ontology [128] as a well-known text representation (WKT) representation on lines 9 to 11.

```
<me> a poso:TrackedFeature, foaf:Person ;
  foaf:name "John Doe"@en ;
  poso:hasPosition <me/position>, <me/approxposition> ;
  poso:hasOrientation <me/orientation> .
<me/position> a poso:AbsolutePosition ;
  rdfs:comment "Absolute position of John Doe"@en ;
  poso:hasAccuracy <me/position/accuracy> .
<me/position/accuracy> a ssns:Accuracy ;
  schema:maxValue "25.0"^^xsd:float ;
  schema:unitCode unit:CentiM .
```

Listing 4.3: Example setup of a tracked person and their properties

```
<position/1654350300000> a sosa:Observation ;
  sosa:hasFeatureOfInterest <me> ;
  sosa:observedProperty  <me/position> ;
  sosa:resultTime "2022-06-04T15:55:00"^^xsd:dateTimeStamp ;
  poso:usedSystem <system/OPS> ;
  sosa:hasResult [ a ogc:Geometry ;
    ogc:hasSpatialAccuracy [ a qudt:QuantityValue ;
      qudt:unit unit:CentiM; qudt:numericValue "9"^^xsd:float];
    ogc:asWKT """
      <http://www.opengis.net/def/crs/OGC/1.3/CRS84>
      Point(4.888028 50.31397)"""^^ogc:wktLiteral ;
    ogc:dimension 2 ] .
```

Listing 4.4: Example observation of the outdoor positioning system

Indoors, our system outputs an absolute Cartesian 3D position as illustrated in Listing 4.5. We identify that the 3D position is made inside a specific deployment on line 8, which contains information about its geometry and the reference system used to convert the coordinates to a common reference frame used by the campus positioning system. The technique used to obtain the result is defined using `sosa:usedProcedure` while the system where this technique is used is defined based on `:usedSystem`.

In the previous example listings, we have shown how a positioning system might model the observations of an absolute position. With the example introduced later in Listing 4.6, we outline how a relative distance to a wireless access point (named `wap_1`) from our `TrackedFeature` can be expressed. Similar to an absolute position, we can have multiple observations of the relative distance. POSO requires the `:isRelativeTo` predicate on a relative position to indicate the feature of interest that the position is relative to. In Figure 4.7, we illustrate how a virtual object is positioned relative to a fiducial marker. The marker itself can be positioned relative to a known coordinate reference system or relative to other objects such as a room or building.

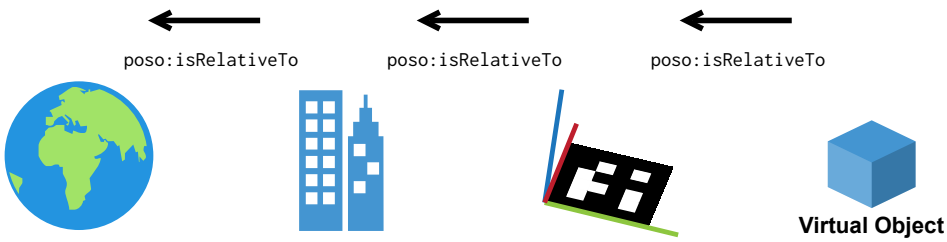


Figure 4.7: Relative positioning of virtual objects and markers

```
<position/1647513000000> a sosa:Observation ;
  sosa:hasFeatureOfInterest <me> ;
  sosa:observedProperty  <me/position> ;
  sosa:resultTime "2022-03-17T11:30:00"^^xsd:dateTimeStamp ;
  sosa:usedProcedure poso-common:KNNFingerprinting ;
  poso:usedSystem <system/IPS> ;
  sosa:hasResult [ a poso:AbsolutePosition ;
    poso:inDeployment <deployment/building_a> ;
    poso:hasAccuracy [ a ssns:Accuracy ;
      schema:maxValue "25.0"^^xsd:float ;
      schema:unitCode unit:CentiM ] ;
    poso:xAxisValue [ a qudt:QuantityValue ;
      qudt:unit unit:M ; qudt:numericValue "5"^^xsd:double ] ;
    poso:yAxisValue [ a qudt:QuantityValue ;
      qudt:unit unit:M ; qudt:numericValue "6"^^xsd:double ] ;
    poso:zAxisValue [ a qudt:QuantityValue ;
      qudt:unit unit:M ; qudt:numericValue "3.5"^^xsd:double]].
```

Listing 4.5: Example observation of the indoor positioning system

Each observable property can also be used to express a fixed result that does not consist of multiple observations. On lines 1 to 5 of Listing 4.6, we utilise this ability to express a fixed result to define the fixed position of a landmark rather than creating a single observation where the position is defined as a result. On lines 10 to 19, we have one observation of this observable relative distance obtained using our indoor positioning system. The result is expressed as a distance using a path loss algorithm and the raw signal strength is expressed in decibel-milliwatts (*dBm*).

To provide a single output for the campus positioning system, we can use the observations from the indoor and outdoor positioning systems shown in Listing 4.5 and Listing 4.4 to compute a fused output based on the weighted accuracy fusion procedure that our campus positioning system implements in Listing 4.2. Using the knowledge about the accuracy, the systems that produced the results and the indoor positioning system deployments, we can perform a fusion with more context than only the self-reported accuracy of each individual subsystem.

4.2.7 Implementation and Technical Evaluation

To test whether our semantic description is feasible to describe a positioning system and its data, we implemented the description in OpenHPS using the `@openhps/rdf` module. This module enables the serialisation and deserialisation of data objects, data frames and other data created and used by a positioning system. In addition, we can also serialise and deserialise positioning models.

```
<landmark/wap_1> a poso:Landmark ;
  rdfs:label "Wireless Access Point 1"@en ;
  poso:hasPosition [ a poso:AbsolutePosition ;
    poso:hasAccuracy [ ... ] ; poso:xAxisValue [ ... ] ;
    poso:yAxisValue [ ... ] ; poso:zAxisValue [ ... ] ] .
<me/position/relative/wap_1> a poso:RelativeDistance ;
  ssn:isPropertyOf <me> ; # Relative distance from <me> ...
  poso:isRelativeTo <landmark/wap_1> ; # to <landmark/wap_1>
  rdfs:comment "Relative position of John Doe to WAP_1"@en .
<position/relative/wap_1/1646891100000> a sosa:Observation ;
  sosa:hasFeatureOfInterest <me>, <landmark/wap_1> ;
  sosa:observedProperty  <me/position/relative/wap_1> ;
  sosa:resultTime "2022-03-10T06:45:00"^^xsd:dateTimeStamp ;
  poso:madeBySystem <system/IPS> ;
  sosa:usedProcedure poso-common:LDPL; # Log-distance path loss
  sosa:hasResult [ a qudt:QuantityValue ;
    qudt:unit unit:Meter ; qudt:value "3.7"^^xsd:double ] ;
  sosa:hasResult [ a qudt:QuantityValue ;
    qudt:unit unit:DeciB_M ; qudt:value "-82"^^xsd:integer ] .
```

Listing 4.6: Example observation of a relative position

All data types defined earlier in Chapter 3 are mapped to the POSO ontology using our Object-Document Mapping (ODM) solution. This enabled us to have a one-to-one mapping of all objects created in OpenHPS to RDF data. However, because the vocabulary is meant to describe positioning systems and their data generically, some adaptations and terminology changes to POSO were necessary to fit our OpenHPS framework.

While OpenHPS primarily uses the POSO ontology for describing the processing of data and systems themselves, other concepts from SOSA and SSN were re-used. These two ontologies provide a solid basis for describing features of interest, sensors and other relevant data. Other ontologies such as M3-lite [203], QUDT [194], Schema.org [204] among many others were used in the mapping. In our RDF module, we provide our vision of mapping concepts from OpenHPS to RDF. However, due to the extensibility of the framework and serialisation of data within this framework, any extension can override this mapping.

Figure 4.8 shows a positioning system, and a single data frame with objects mapped to the SOSA, SSN and POSO ontologies. Similar to the positioning system representation illustrated in Figure 4.3, the positioning system consisting of sensors, algorithms and output data can be represented as a graph network.

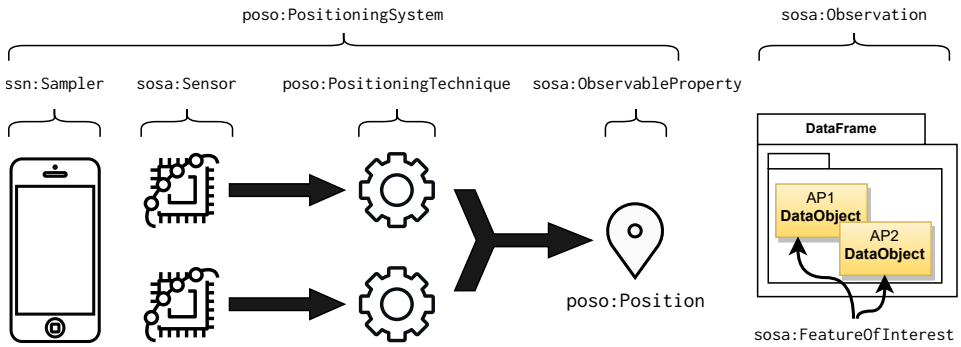


Figure 4.8: Mapping of OpenHPS concepts (sensors, graph nodes and data) to the SOSA, SSN and POSO ontologies

Graph Nodes and Edges

A main design choice in OpenHPS was the graph-based processing through nodes and edges. The OpenHPS framework uses source nodes, sink nodes and processing nodes. In POSO, source, sink and processing nodes are mapped to `sosa:Procedures`. These procedures describe the input, output and processing itself, allowing the chaining of procedures similar to the graph-based processing of OpenHPS. Edges in OpenHPS are mapped to the relations for the input and output of each procedure.

In the case of source nodes, the `sosa:Procedure` is the act of *producing data*, while on processing nodes it is the `sosa:Procedure` of *manipulating the data*. That is, the sensor producing data is a procedure that has its intrinsic functionalities, such as the throughput and output type.

Built-in OpenHPS nodes are described online using the procedure types defined in POSO. This provides additional context beyond simply specifying a particular algorithm used to process information. This description is automatically generated using our RDF module. As an example, the multilateration procedure that is used in OpenHPS is described and identified using the following URI: `https://openhps.org/terms/procedure/MultilaterationNode.ttl`. By using these URIs, third parties do not only know that a `:Multilateration` algorithm was used, but also which implementation; in this case, the implementation of the OpenHPS framework.

Data Objects

As detailed in Chapter 3, all spatial objects in OpenHPS are considered data objects. A data object can represent a person, a smartphone owned by a person or even an environment. In POSO, each of these objects can be mapped to a

`sosa:FeatureOfInterest` with an optional `ogc:SpatialObject` type from the GeoSPARQL vocabulary.

In the case of spatial objects with an enclosing object, the `ogc:sfWithin` predicate is used to indicate that an object is within another object. Enclosing objects are often used to indicate that a sensor is part of a smartphone, or to indicate that an environment is within another environment, in which case both are spatial objects.

Data Frames

Data frames were described in Section 3.4.6 as momentary snapshots of information. POSO does not offer the concept of data frames in its vocabulary but treats all information within a data frame as an `sosa:Observation`. In practice, this entails that all data within a data frame is now treated as individual observations of a set of data objects. No information is lost in this way.

4.2.8 Conclusion

In this section, we introduced our generic positioning system ontology, POSO, to describe concepts relevant to a positioning system. These concepts include the different observable properties that can be obtained by a positioning system, the different categories of systems and the different algorithms and techniques these systems can implement to handle positioning. Our generic positioning system ontology not only focuses on common geospatial and geographical concepts that are already described in various existing vocabularies but also offers a novel vocabulary for describing generic data produced by a positioning system. We expanded the SSN and SOSA ontologies by providing common procedures and observable properties. By further presenting the `poso-common` module, we illustrated how POSO can be expanded with a set of common algorithms, existing systems and platforms. Finally, we illustrated the usage of POSO with a scenario containing two positioning systems and a hybrid positioning system using a high-level fusion technique. In this demonstration, we have shown how each positioning system might be modelled using POSO and how observational data can be expressed.

Future work will focus on adding additional positioning techniques and algorithm procedures, further describing the input and output that each procedure provides. By using known input and output RDF shapes that are used in different positioning systems, we can further classify a positioning system's technologies and the output they provide. While we already offer procedures for obtaining map information (i.e., Simultaneous Localisation and Mapping), our vocabulary is not aimed at describing geospatial data.

4.3 Object-Document Mapping for Semantic Data

In the previous section, we have detailed our POSO ontology for describing positioning systems. This ontology was partly based on the OpenHPS framework, which defines the pipeline and data structure of a wide range of positioning systems. To validate that we can describe all concepts from OpenHPS with POSO and other ontologies, we needed an extensible solution on top of OpenHPS to handle this mapping.

Object-Document Mapping (ODM) is a tool for developers to automatically map objects and fields to document storage [205]. To easily utilise linked data in our OpenHPS framework, we developed an ODM solution on top of our framework to bridge the gap between research questions RQ1.1 and RQ1.2 (that generalise the input and output of a positioning system) to research question RQ2 (that aims to answer the interoperability of a positioning system). To create and design an interoperable positioning system, defined as a system with generalised input and output data, we needed a generalised method to easily map this input and output data to an interoperable format.

Our OpenHPS framework already offers the serialisation of objects to JSON data using a modified TypeJSON implementation. The modification we made to TypedJSON, detailed in Section 3.6, enabled us to offer modular serialisation to data formats other than JSON. We have expanded OpenHPS with the `@openhps/rdf`[4] module, enabling the serialisation and deserialisation of objects to RDF data. In addition, we provide a *service driver* in OpenHPS that enables the querying of these RDF resources by writing queries in MongoDB[5] format, similar to the queries used to query JSON data. This enables developers to easily implement complex queries on semantic data stored in RDF format without modifying the queries.

The ODM solution we developed is tailored to the OpenHPS framework with existing mappings to our own ontologies we have developed for aiding in the mapping of positioning systems and the data they produce. In the future, this solution can be decoupled from OpenHPS to provide developers with a generic tool.

Frameworks such as Soukai [206] (with the Solid driver), Linked Data Objects [207] and KOMMA [208] offer serialisation and deserialisation of objects to Solid Pods, but do not handle the (SPARQL) querying of data and are not feasible to deserialise polymorphic data or RDF structures that do not follow the same of the serialisation. These limitations hinder the ability to easily interact with semantic data in a meaningful and interoperable way.

[4] `https://openhps.org/docs/rdf/`
[5] `https://www.mongodb.com/docs/manual/tutorial/query-documents/`

Our tool consists of three main stages: (stage 1) manual mapping of objects and fields to RDF, (stage 2) automatic serialisation of objects using the mapping and (stage 3) the construction of queries using the mapping. To aid in the mapping, we provide variables for URIs of classes, properties and individuals from popular ontologies relevant to positioning systems. These variables are automatically generated and are further detailed in Section 4.3.1.

Next, we provide developers with *decorators* which provide metadata at runtime about classes, fields and methods. This metadata provides our ODM solution with information on how to serialise or deserialise classes and fields to RDF. Using the metadata provided by developers using the decorators, we keep a changelog of changes made to objects at runtime. Our solution uses this changelog to serialise and deserialise data objects and data frames in OpenHPS to RDF.

Last but not least, while serialisation and deserialisation are the primary goals of our object document mapper, OpenHPS requires the ability to query serialised data. Therefore, we integrated a solution that converts a traditional query from any OpenHPS data object service to a SPARQL query. This querying is further detailed in Section 4.3.7 along with several examples.

4.3.1 Namespace Generation

Based on the `rdf-namespaces` tool by Tunru [209], `@openhps/rdf` comes with a utility to fetch ontologies and vocabularies from local and online sources and generates variables to easily use RDF types, predicates and individuals without manually specifying the URIs.

Listing 4.7 shows a code snippet of such TypeScript vocabulary generated automatically for one of our own ontologies using a modified version of the tool by Tunru [209]. This provides developers with easy-to-use variables for RDF types or predicates, while also providing developers with documentation about the purpose of these types or predicates.

4.3.2 Class and Field Decorators

Similar to other ODM solutions, our object document mapping solution to RDF makes use of class and field *decorators*. These decorators provide metadata to classes, fields or methods at runtime. At runtime, these decorators create a variable within the *prototype* of the class in which they are used, containing information about all the fields and their serialisation options, as well as information about any class that extends upon this class.

Figure 4.9 illustrates access to the prototype in JavaScript. The prototype of a `Person` class can be obtained using `Person.prototype`. This prototype object

```typescript
type IriString = `${'http' | 'https'}://${string}`;
type OwlObjectProperty = IriString;

/**
 * angle
 * Quantitative angle result value for axis-angle ...
 * http://purl.org/poso/angle
 */
export const angle: OwlObjectProperty =
  'http://purl.org/poso/angle';

/**
 * y-axis value
 * Quantitative result value along the Y-axis ...
 * http://purl.org/poso/yAxisValue
 */
export const yAxisValue: OwlObjectProperty =
  'http://purl.org/poso/yAxisValue';
```

Listing 4.7: Snippet from an automatically generated TypeScript vocabulary of the POSO ontology based on the `rdf-namespaces` tool [209]

contains a unique variable that links to a metadata object containing information needed for the serialisation and deserialisation of instances of the `Person` class.

In OpenHPS, we provide two main decorators: (1) a `SerializableObject` decorator to describe an object that can have multiple instances (i.e., individuals in RDF) and (2) a `SerializableMember` decorator to describe the predicates of these individuals. The latter has several sub-types for indicating an array, set or map of elements. Behind the scenes, these decorators extend the existing decorators used by the TypedJSON framework, with additions for better handling of option inheritance. In addition, we also add an extra `rdf` option to configure RDF-specific serialisation and deserialisation information.

Listing 4.8 provides a basic example of a `Person` object that contains a first name and family name as variables. The class is decorated to be mapped to a `foaf:Person` type in RDF. As an example, we provide a custom `serializer` function that sets the URI to the combination of the first name and family name of the person, separated with an underscore.

Member Decorators

Each data member in a class that should be serialised to RDF can have a decorator to indicate the serialisation method and properties. First and foremost, this includes

[6] `https://stackoverflow.com/q/572897/`

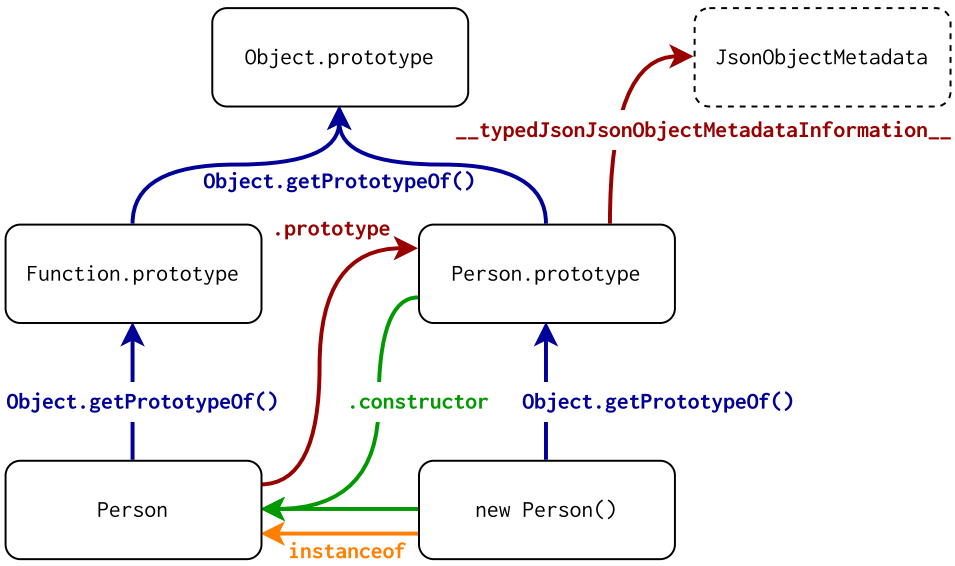


Figure 4.9: JavaScript class prototype and TypedJSON metadata[6]

the predicate URI(s) that can be used for serialisation and deserialisation. For deserialisation, we also provide the option to either define a custom `deserializer` function or a SPARQL query that is executed on the connected triples of the current subject.

As an example, Listing 4.9 shows the data members of the `Person` object (i.e, the first name and last name) being decorated to specify the predicates for serialisation to RDF. Using these member decorators, developers can easily define how each data member should be represented in RDF. By default, only the first predicate is used for serialisation to prevent redundant information. However, other than during

```
@SerializableObject({
  rdf: {
    type: foaf.Person,
    serializer: (obj: Person, baseUri?: string) => {
      return { value: `${obj.firstName}_${obj.familyName}` }
    }
  }
})
export class Person extends DataObject {
  // ...
}
```

Listing 4.8: Person object decorated with RDF serialisation options

```
@SerializableObject(
  // ...
)
export class Person extends DataObject {
  @SerializableMember({
    rdf: { predicate: [foaf.givenname, foaf.firstName] },
  })
  firstName: string;
  @SerializableMember({
    rdf: { predicate: [foaf.surname, foaf.familyName] },
  })
  lastName: string;
}
```

Listing 4.9: Person object data members decorated to provide RDF serialisation and deserialisation

the serialisation, when multiple predicates are defined, they will all be considered during the deserialisation process.

4.3.3 Object Change Log

Similar to the implementation of the Solid client by Inrupt[7], a change log has to be maintained that keeps track of all *triples* that are deleted, modified or added. In a relational database or document storage, a change log is only required to keep track of the updated attributes and rows. However, in a graph structure, changes need to be explicitly monitored and managed to be able to delete the triples that were modified.

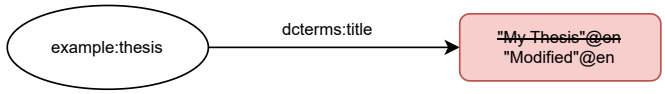


Figure 4.10: Detecting changes in triple ibhects

Figure 4.10 illustrates the importance of monitoring changes in RDF. When the name of an object is updated from “My Thesis” to “Modified”, the change includes two triple changes: (1) the triple `example:thesis dcterms:title "My Thesis"@en .` should be removed, (2) a new triple should be created that links `example:thesis` to the new title. Similarly, changes to a predicate require the deletion and addition of triples to ensure that the RDF graph accurately reflects the updated information. By maintaining an object change log, developers can efficiently track and manage these changes in RDF data.

[7] https://github.com/inrupt/solid-client-js

Unlike Inrupt's Solid client, which keeps track of changes by explicitly indicating the addition and removal of predicates, our solution keeps an internal change log within each class itself. When manipulating the fields within these classes, these changes are added to our internal changelog[8].

4.3.4 Serialisation

Serialisation of objects to RDF is relatively straightforward. The decorators ensure that the object properties are directly mapped to RDF in the shape that the developer foresees in their application. Custom serialisation functions can be defined on objects and properties to modify the output, but in general, the JavaScript variables with type definitions from TypeScript are mapped to objects, literals and blank nodes.

Compared to other frameworks such as the Solid driver in Soukai [206], Linked Data Objects (LDO) [207] and KOMMA [208], our solution offers a more simplified approach to serialising data. In LDO, developers have to explicitly define which fields and objects should be serialised to which predicates, which offers a lot of flexibility but can be time-consuming and error-prone. Soukai is a generic ODM library that is not specific to linked data. One of the *engines* that enables Soukai to directly store data in a Solid pod requires developers to define *model* schemas. Other than our tool, these stored models are specific to Soukai and do not work by annotating existing fields and objects with decorators. Models have to follow the exact schema for them to be deserialised.

Finally, KOMMA is a Java framework for serialising data to RDF. Similar to our framework, it works by annotating classes and fields using decorators. However, it does not support querying the data or deserialising data based on queries. Furthermore, it only offers one-to-one serialisation of classes to RDF and does not tackle more advanced serialisation where the data of objects may need to be transformed.

Object Types

Similar to the serialisation of member types, classes can be decorated using a `@SerializableObject` decorator where one or more RDF types can be assigned. While member decorators indicate the predicates used to link a subject to an object, the class decorators indicate the `rdf:type` used to serialise an instance of a class. During the deserialisation, all these types will be considered for deserialisation. However, only the first type is used for serialisation to prevent a class from being serialised to a lot of redundant subclasses of each other.

Inheritance of classes, as depicted in Figure 4.11, entails the extension of classes. Each of these classes can have a set of RDF types that are used during serialisation.

[8] In JavaScript this is accomplished by wrapping these models in a proxy catching each method invocation and field access

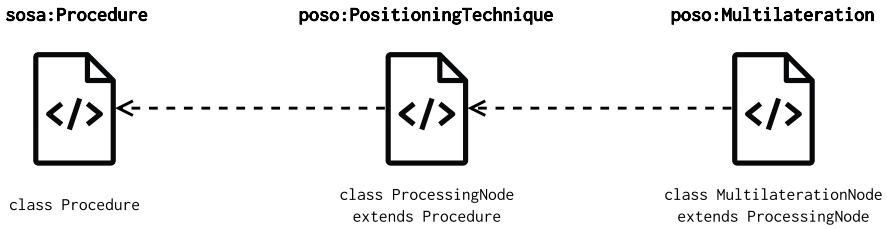


Figure 4.11: Inheritance of classes with RDF types

Data Types

The following variable types are mapped:

- `String`: A string in RDF contains a language tag to indicate the language of the string. Serialisation options enable developers to indicate if a string variable is represented in a specific language.
- `Boolean`: Mapping of boolean variables does not involve any additional complexity. Boolean variables are directly mapped to RDF literals with the appropriate data type.
- `Number`: In JavaScript (and TypeScript), all numeric values are represented as a simple number. They can contain an integer or decimal number. However, in RDF, number types need to be distinguished. A TypeScript number is serialised in RDF to a double literal by default, which can be used to contain both integers as well as decimal numbers. Developers can specify the precision of the number in the member decorator to modify this default behaviour. This allows developers to explicitly indicate if a number represents an integer, float or any other available numeric type.
- `Date`: A date object is converted to an `xsd:date` or `xsd:dateTime` depending on the serialisation options.
- `Array`, `Set` and `Map`: In TypeScript, field members are explicitly typed to be either atomic values, arrays, sets or maps. When serialising to RDF, cardinality is not enforced through types. In other words, in RDF, a simple `name` of a person could exist multiple times for one subject by having multiple triples with a `foaf:name` predicate. Instead, data shapes and OWL constraints can be used to indicate how many values a property can have.

Predicates

Members that are decorated with predicates will be used to indicate the relationship between the subject and other objects or atomic values. Multiple predicates can be defined for a single field. Similar to the RDF type, only the first predicate is used

to serialise the data. However, all predicates will be considered when attempting to deserialise the data.

4.3.5 Deserialisation

One of the challenges in object-document mapping for RDF data is the deserialisation from RDF data to objects due to the complex and open structure of linked data. To solve this issue, our deserialisation solution combines the `rdf:type`(s) of individuals together with the available predicates an individual has.

Object Type Mapping

Initialising the deserialised object requires the deserialisation of the type of the object. Internally, a dictionary is kept that tracks which TypeScript classes are serialised to which RDF types. Developers can manually add new records to this dictionary based on their requirements, enabling them to specify additional types for deserialisation.

1. A selection of TypeScript classes is created based on the `rdf:type`(s) of the individual
2. When more than one possible class is found, a positive weight is assigned to each possible TypeScript class that contains an expected predicate that would have been added during the serialisation. A negative weight is added for each predicate that is not expected and will not be deserialised
3. Based on the weights, the deserialisation algorithm selects the class with the highest overall weight as the most likely match for the individual being deserialised

Data Members

After determining the type of class, we can fetch the data members that should be deserialised. For each data member of a class, the deserialisation process involves matching the predicates in the RDF data to the properties of the object. This is done based on the decorators assigned to the properties during serialisation. If a predicate in the RDF data matches a property of the object, the value of that predicate is assigned to the property. Additionally, handling complex data structures like arrays or nested objects requires special consideration during deserialisation.

Remaining Predicates

In the deserialisation process, any remaining predicates that were not mapped to data members in the class are stored as key-value pairs in a hidden variable within the class. This ensures that no data is lost during the deserialisation process and provides flexibility for handling unexpected or additional data.

During the (re)serialisation of these classes, these unmapped predicates are included in the serialisation to ensure that the serialisation output is the same as the deserialisation input. Developers can also access the unmapped predicates within custom serialisation and deserialisation functions.

4.3.6 Data Shapes

Basic deserialisation expects the data to be structured based on the decorators. With interoperability in mind, this assumption does not hold when multiple applications store their data in different formats. One possible solution that frameworks such as Soukai [206] use to mitigate this is to make use of data shapes. Data shapes define the structure of data using a set of vocabularies. These shapes can be used to standardise the way serialisation of data occurs, while also enabling applications to provide other parties that can access the data with a *blueprint* on how they store RDF data.

Our ODM solution that is built into OpenHPS currently does not support the loading of data shapes for manipulating how data is deserialised or re-serialised, but it supports the generation of Shapes Constraint Language (SHACL) shapes for known data types that can be stored alongside the data.

4.3.7 Querying

To facilitate querying of RDF data within OpenHPS, we have implemented a query construction driver that is used in a data object service. This driver allows developers to write queries in MongoDB syntax similar to how querying currently works in OpenHPS. The MongoDB syntax was chosen due to its flexibility and ease of use in querying JSON-like data structures. More information about the syntax can be found in Section 3.8.1.

With the `@openhps/rdf` module, developers can construct queries that include conditions and aggregation functions. These queries can then be executed on the RDF data stored within the OpenHPS framework. This allows for a seamless choice between in-memory, MongoDB (using the `@openhps/mongodb` module) or RDF data storage without interference from the developer in both storage and querying. Each query is constructed with a similar syntax to the one used in MongoDB and is converted to an SPARQL query that is executed on the local or remote storage.

Listing 4.10 is the template for a SPARQL CONSTRUCT query [210] that outputs a graph based on the conditions provided in the WHERE clause. This query constructs a graph with the subject, its properties, and all its children along with their properties. Lines 5 and 6 ensure that all the child predicates are selected by specifying the predicate of the subject to *be connected* or *not be connected* by the dummy `example:overrides` predicate. This essentially allows for a recursive selection of linked objects, as it will select all objects that are connected via a

```
CONSTRUCT {
  ?subject ?prop ?val.
  ?child ?childProp ?childPropVal.
} WHERE {
  ?subject ?prop ?val;
    ((example:overrides|!example:overrides)+) ?child.
  ?child ?childProp ?childPropVal.
  # Conditions using ?subject and ?child here
}
```

Listing 4.10: Generated query for outputting subjects and all their children

predicate. Line 8 would contain the conditions from the transformed MongoDB queries in subsequent examples.

```
                                displayName: "Maxim"
{
    { ?subject a sosa:FeatureOfInterest. }
    UNION
    { ?subject a ogc:SpatialObject. }
}
?subject rdfs:label "Maxim".
```

Listing 4.11: Generated query to select all triples pertaining to a data object with the name (i.e., `rdfs:label`) "Maxim"

Using this template, Listing 4.11 demonstrates the condition generated from a query that specifies the display name of a data object. The MongoDB query is displayed in the top right of the listing and represents the query that the developer used within the OpenHPS framework. The filter criteria searches for subjects with a specific data type (i.e., either a `sosa:FeatureOfInterest` or `ogc:SpatialObject`). These data types were not inputted by the user, instead, they were extracted from the `@SerializableObject` decorator that specified which RDF types should be considered to deserialise a `DataObject`. Next, since our search criteria includes the display name of a data object, an additional condition is added that specifies that the subject should have an `rdfs:label` that matches `"Maxim"`. More precise filtering is done during the deserialisation phase, where the returned predicates are also taken into consideration to determine the relevant data types. This allows the deserialisation to match the output to the correct subtype (e.g., a `SensorObject`) rather than deserialising to a generic `DataObject`. However, by introducing a conditional filter in the query itself, we can filter the data earlier in the processing.

When a query includes a specifier for the subject URI, the SPARQL query generator will ensure the appropriate filtering is included as shown in Listing 4.12. The

```
uid: "mvdewync"
FILTER(?subject = :mvdewync)
```

Listing 4.12: Generated query to select all triples pertaining to a data object with the uri :mvdewync

subject is filtered to only include the subject with the URI `:mvdewync`. Unlike Listing 4.11, which includes an additional condition for filtering the type, the SPARQL query generator will omit this filter due to the existence of a filter on the URI, which already narrows down the results to a specific subject. Similar to the previous example, the deserialisation phase can still determine the correct type used for deserialising the data.

```
position: { x: 10, y: 20 }
{
    { ?subject a sosa:FeatureOfInterest. }
    UNION
    { ?subject a ogc:SpatialObject. }
}
{
    {
        ?subject <http://purl.org/poso/hasPosition> ?o1.
        ?o1 <http://purl.org/poso/xAxisValue> 10 .
        ?o1 <http://purl.org/poso/yAxisValue> 20 .
    }
}
```

Listing 4.13: Generated query to select all triples pertaining to a data object where the position has an X-value of 10 and Y-value of 20

Queries where the condition depends on an inner object are also supported. With Listing 4.13 we illustrate a SPARQL query that is generated to retrieve data objects with a condition based on the X and Y value of their position. In our query, we specify that the X value of a position should be 10, while the Y value should be 20. The query generator first constructs a triple pattern to select the position on line 15 with the variable name `?o1`. Next, the predicates for the X and Y values are determined and use the `?o1` variable as the subject to narrow down the result. The query generator will assign a new variable name for every inner object that has a condition. In case the query depends on the condition of two separate inner objects (e.g., the position and orientation), a 2nd variable called `?o2` would be used.

Similarly to Listing 4.11 that selects data objects based on the display name, Listing 4.14 demonstrates how a regular expression is converted to a SPARQL query. The REGEX function in SPARQL is used to test the regular expression against the object.

```
displayName: /Maxim/g
```

```
{
    { ?subject a sosa:FeatureOfInterest. }
    UNION
    { ?subject a ogc:SpatialObject. }
}
{
    ?subject rdfs:label ?object.
    FILTER(REGEX(?object, "Maxim", "i"))
}
```

Listing 4.14: Generated query to select all triples pertaining to a data object with a name that matches a regular expression

Filter Conditions

MongoDB querying supports specifiers to indicate if a value should be greater than or less than a specific value. In our query generator, we also support these conditions via SPARQL queries. The following filter conditions are supported: greater than (`$gt`), less than (`$lt`), greater than or equal (`$gte`) and less than or equal (`$lte`).

```
"position.x": { $gt: 10, $lt: 20 }
```

```
{
    { ?subject a sosa:FeatureOfInterest. }
    UNION
    { ?subject a ogc:SpatialObject. }
}
{
    {
        ?subject <http://purl.org/poso/hasPosition> ?o1.
        {
            ?o1 <http://purl.org/poso/xAxisValue> ?o2.
            FILTER(?o2 > 10) FILTER(?o2 < 20)
        }
    }
}
```

Listing 4.15: Generated query to select all triples pertaining to a data object where the position has an X-value of strict more than 10 strict less than 20

In Listing 4.15, a query is generated to find all data objects with a position where the X-axis value is strictly greater than 10 and strictly less than 20. Similar to Listing 4.13, a variable named `?o1` is created for the position of the subject. Next, a variable is created for the X value (`?o2`) after which two filters are created to ensure this variable is within the specified boundaries. The query that is generated

could have been written without the inner group graph pattern for filtering `?o1`. However, our ODM solution will automatically put all individual conditions into a group pattern to facilitate the generation process.

Optional Conditions

An optional condition is a condition that does not have to be met for the query to return results. It provides flexibility in the search criteria. In MongoDB, this optional condition is created using the `$or` syntax.

```
$or: [{ displayName:
"Maxim" }, {
displayName: "Beat" }]
```

```
{
    { ?subject a sosa:FeatureOfInterest. }
    UNION
    { ?subject a ogc:SpatialObject. }
}
{ ?subject rdfs:label "Maxim". }
UNION
{ ?subject rdfs:label "Beat". }
```

Listing 4.16: Generated query to select all subjects that either have the display name “Maxim” or “Beat”

As shown in Listing 4.16, an optional (OR) condition is translated as a `UNION` between conditions. In the given example, all subjects are selected that either have the name “Maxim” or the name “Beat”. More elaborate queries can contain nested optional conditions.

4.3.8 Conclusion

Using the object document mapping tool created in the `@openhps/rdf`, we facilitated the development of a positioning system that can store and retrieve data in RDF. Our tool expands on the decorators provided by our OpenHPS framework to help developers provide additional metadata on how objects and fields are mapped to RDF. We designed a solution that can translate the existing MongoDB query format used by OpenHPS to SPARQL queries.

While our ODM solution offers a simplified way to serialise, deserialise and query RDF data, it still has several limitations. In its current state, the ordering and sorting of data is handled in memory rather than by the SPARQL engine itself, which may result in performance issues for large datasets.

Another limitation in the current implementation of our tool is the limited support for array operations. MongoDB queries support various array selection criteria. In our current implementation, these are handled in memory rather than transforming them into SPARQL queries.

While our ODM solution is tailored to the OpenHPS framework and the data structure it produces, future work will separate the ODM solution and logic from the OpenHPS framework. This separation will benefit other developers as well as the solution itself.

4.4 User-centric Storage Using Solid

In Section 4.2, we discussed our POSO ontology to allow the semantic description of positioning systems and the data they produce. Interoperability also entails that data is accessible by other services. To enable this access, we propose a solution for user-centric storage of positioning system data, where the subject that is being tracked manages the data required and computed by each positioning system that tracks this subject. The user-centric storage enables the *decentralisation* of positioning systems, as these systems can collaborate to track an individual.

4.4.1 Architecture

Our proposed solution architecture is based on the Solid project by Tim Berners-Lee [139]. Solid offers personal data vaults called *Pods*, storing a user's data. Users can choose the *provider* storing their Pod and thereby determine which organisation to trust storing their data. Applications and websites that want to store or fetch a user's data can ask the user to authenticate themselves to their Pod provider, giving those applications access to parts of the vault where they can create or read data. The use of Solid Pods enables users to put their data on the decentralised web. Users can have multiple personal data vaults with one or more storage providers.

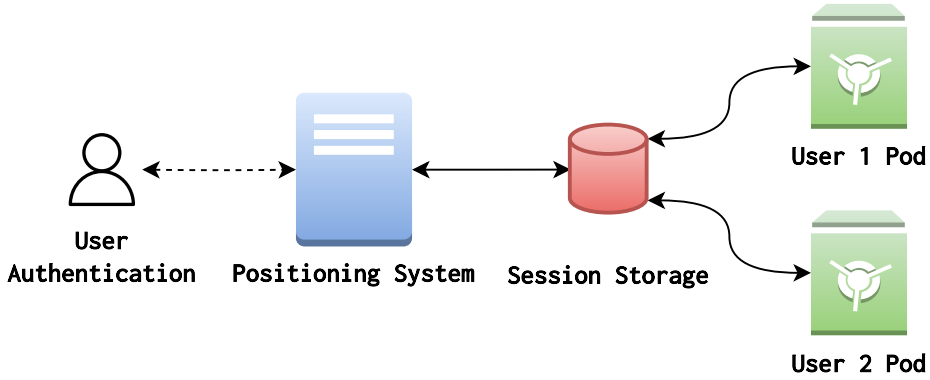


Figure 4.12: Basic architecture of linking a positioning system with Solid

In Figure 4.12, we show the basic architecture of a positioning system using Solid. Users authenticate through a user interface (i.e., either a website or a smartphone application). The authentication process in this system provides a session key

that can be used to access the user's private Pod. Each user will have their own application-specific unique session key stored in the session database, enabling access to their Pod. These keys are the only data that the system has to store locally.

Once the positioning system has access to a user's private Pod, the Pod will be used to store all relevant tracking data. This might range from processed data, such as a user's geographical position, to raw sensor data that should be persisted. While the positioning system can still store some, if not all, data on privately owned servers, the philosophy of Solid is that this would be governed by laws such as the European Digital Markets Act (DMA) [211]. Similar to the governing of the usage of data vaults, applications with read access to a Pod can theoretically clone the data locally. However, laws like the European General Data Protection Regulation (GDPR) [9] aim to protect users from such practices.

The user has control and transparency over the data that is being created or updated by a positioning system, as well as the ability to revoke future access to this data at any time. Compared to the three-layer model of a Location-based Service (LBS) [63], our approach moves the responsibility of the reasoning middleware layer to the application that reads the data, but with additional semantic rules and knowledge on how this reasoning can be performed. This offers a lot of flexibility, making the personal data vault not simply an LBS, but also a service for raw unprocessed data.

While personal data vaults enable users to revoke access to specific data at any time, these systems may still have collected data from before access was revoked. In this case, regulations like the GDPR provide users with the right to request the deletion of all their personal data held by a service provider. In Section 7.6.5 of our integrated solution, we provide more details on how users can be made aware of the data collection of a positioning system.

Location data should be interoperable between different systems, meaning that data created by an indoor positioning system in one building should be compatible with another positioning system or application. This allows our positioning data to be processed by different positioning systems to enable the handover of tracking [212], or allow multiple indoor positioning systems to be linked in one non-proprietary smartphone application. Linking a positioning system to a user entails that the user provides the system with access to their data. Interoperability on this scale further enables the decoupling of positioning systems from the user interfaces and applications that generate and use the data.

Figure 4.13 shows an architecture where a user authenticates to two individual positioning systems that work independently from each other. Both systems can access the same data vaults. In addition, we have a navigation application that uses the data stored in the decentralised Pods as a location-based service. Alternatively,

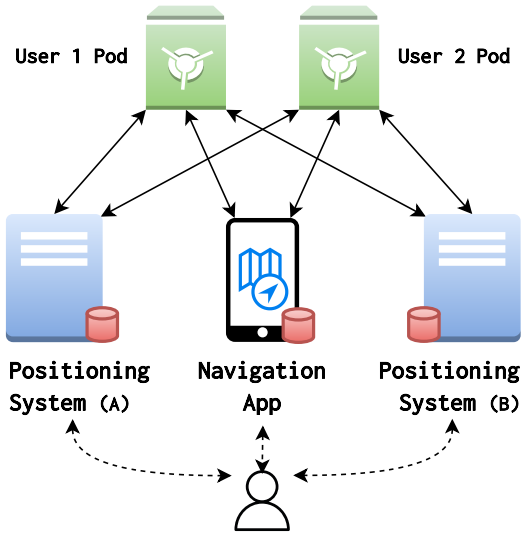


Figure 4.13: Two positioning systems using same user Pods

a single application could be responsible for obtaining sensor data on a smartphone that is then being processed by a positioning system.

Solid allows the storage of regular documents such as media files as well as linked data [124]. We use the POSO ontology to describe the setup of a positioning system and its algorithms, the output data with different granularities and a vocabulary that provides additional context on the accuracy of each output position. Figure 4.14 showcases an example structure of a Pod, consisting of a profile card, and a set of properties containing observations for those properties.

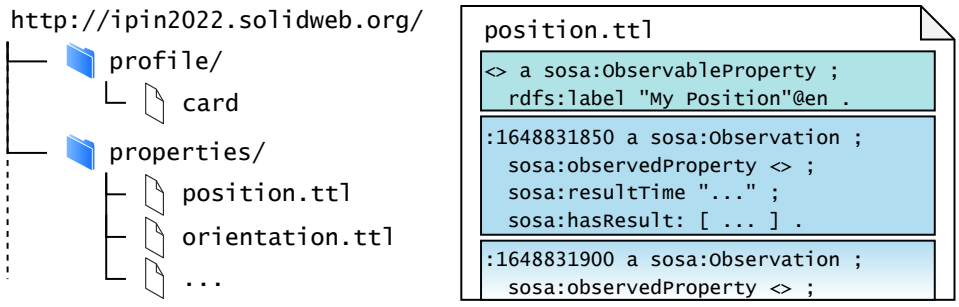


Figure 4.14: Overview of a fictional user's data stored in a Solid Pod

Linking a user to a positioning system is the first step towards user-centric data storage for positioning systems. However, internally, a positioning system often needs to track multiple users, each with their own data. To help manage this ownership

in the OpenHPS framework, we link all data using the Web Identifier (WebID) of the user. A technical example is provided in Appendix A.3.4.

4.4.2 Solid Pod Properties

In our paper *A Solid-based Architecture for Decentralised Interoperable Location Data* [43] we proposed a solution to store location data such as orientation, position and velocity as individual *properties* that can be linked to a user whom these properties belong to. We aimed for an implementation that allows multiple *observations* for a single property, allowing the storage of a trajectory and historical positioning data.

Slabbinck et al. [213] indicated in their work that the use of single LDP resources [214] for different *versions* of a position requires proprietary applications, which is not beneficial for obtaining interoperability between multiple applications or positioning systems. In addition, they rightfully argue that updating the same resource is not feasible for fast-moving data, which is generally the case when working with sensor data or computer position data. With our current solution, the resource files would rapidly become very large.

In their work, Slabbinck et al. [213] propose a solution for *Linked Data Event Streams* [215] within Solid Pods. A new container is created for each *granular version* of a position, along with its access control rights. When a resource becomes too large, a new resource is created where new data will be stored. This solution facilitates client-side querying as resources remain manageable in size. However, it does complicate the storage of new data, as it must adhere to the LDES specification to structure the data.

4.4.3 Linked Data Event Streams

One of the technical challenges involved with storing large amounts of data in Solid is the need for efficient ways to handle streams of data. In an RDF database with SPARQL endpoint, triples can be added or queried efficiently from a dataset. However, in Solid, RDF data is stored in resources that become larger over time as more data is added. This not only increases the complexity of making changes to this data, but due to the lack of a SPARQL endpoint, it also makes it challenging to efficiently retrieve part of the data.

Linked Data Event Streams (LDES) offers a method to create and, more importantly, describe immutable event streams [215]. As a core ontology, LDES uses the TREE hypermedia specification [216], which structures the data over multiple smaller fragments, which are optimally structured in a hierarchical tree structure. Splitting the data over multiple fragments provides greater flexibility in storing and retrieving chunks of data. Since data produced by sensors or our positioning

systems is a time series, fragmenting the data based on time intervals can greatly improve the efficiency of data retrieval.

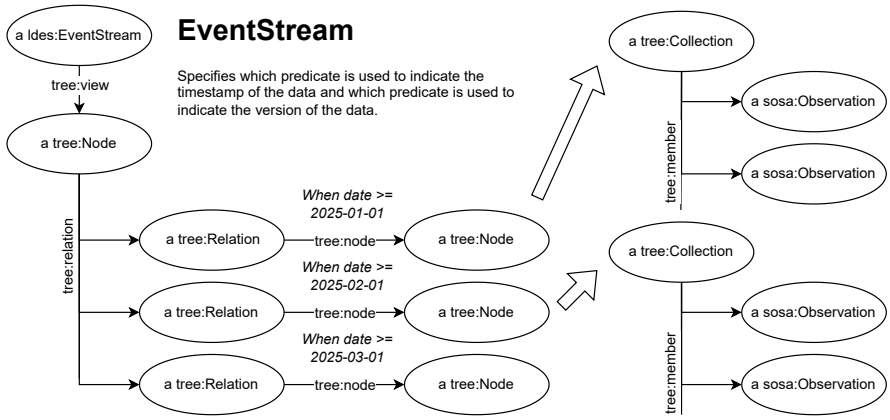


Figure 4.15: Linked Data Event Streams example with multiple TREE nodes and observations

The TREE specification consists of three main concepts; a `tree:Collection` describing the data and containing (part of) the data within the collection, a `tree:Node` that describes a fragment of the collection and finally `tree:Relation`s that describe how nodes are related to each other. Various relations are defined in the specification, such as simple "greater than" or "less than" relations as well as more complex geospatial relations. Together, this creates linked data fragments (LDF) of the collection, spread over multiple *nodes* [217].

LDES builds on top of the TREE specification by defining an event stream as an immutable collection that can only be appended. It enforces a stream as a time-based source of information and adds a vocabulary to indicate the retention policy of data stored in this stream as well as the strategy for splitting up the stream into buckets.

Figure 4.15 illustrates the basic concept of an event stream created using LDES and TREE. An event stream defines the root *node* of the data structure. This root node can indicate relations to other nodes that contain information about the `tree:Collection`. In our adapted solution for providing user-centric storage to data produced by positioning systems, we continue with our previous approach of using Solid to store the data. However, to ensure the scalability of the data contained within Solid, we utilise LDES to structure the stream of information.

While LDES is inherently a Linked Data vocabulary, it has found its way to database management systems such as PostgreSQL and MongoDB with the Vlaamse

Smart Data Space (VSDS) project[9]. This enables developers to utilise the semantic description of LDES while simultaneously leveraging the performance and scalability of traditional database systems needed for handling real-time streaming data.

4.4.4 Communication Broker

In the context of our proposed architecture for user-centric interoperable location data storage, the integration of Linked Data Event Streams (LDES) provides a robust solution for handling large data streams efficiently within Solid Pods. However, in a real-time system, a method of communication is needed to ensure that new data is processed and stored when it is made available.

Solid storage providers implement the Linked Data Notifications (LDN) specification [218], which allows applications to listen for changes made to an individual resource or all resources within an LDP container. In our work [44] and [42], we propose two independent solutions for utilising these notifications to offer real-time data processing. In our first solution [44], we created new `schema:Actions`[10] items in a container to act as events. In the second solution [42], we used these actions for real-time manipulations in a 3D environment.

4.4.5 Adapted Solution

We developed the `@openhps/solid`[11] module for OpenHPS, which enables the use of Solid to store data using Linked Data Event Streams and individual observations. The solution extends our earlier prototype of storing properties and observations in a Solid Pod. However, to optimise the storage, the structuring is based on the work of Slabbinck et al. [213], but with a more specific focus on structuring properties.

As detailed earlier in Section 4.4.2, properties represent all *observational* properties of an object, or in our user-centric use-case, a user. Each property consists of a description and a collection of observations for these properties.

Fetching Properties

When a user authenticates a positioning system with their Solid Pod, the system should first determine the user's existing properties. While our implementation will store properties in the `/properties` container relative to the root of the Solid storage, existing properties in the Pod do not need to adhere to this data structure.

The profile card of the user who owns the Solid storage should contain a reference to all properties about this individual. Figure 4.16 illustrates the discovery of

[9] `https://github.com/Informatievlaanderen/VSDS-LDESServer4J/`
[10] `https://schema.org/Action`
[11] `https://openhps.org/docs/solid/`

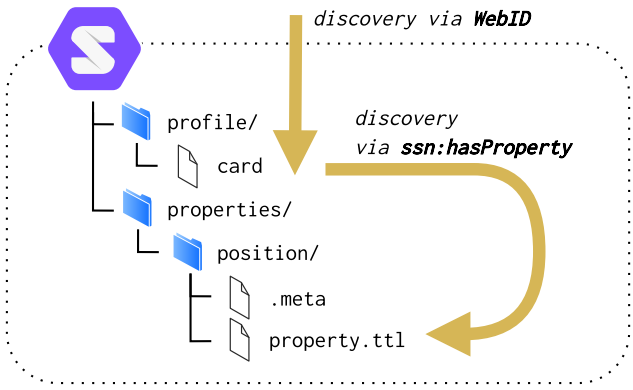


Figure 4.16: Discovery of properties from the user

properties within a user's Pod. A user authenticates their Pod with a positioning system along with its associated Web Identifier. This WebID links to the profile card within the Pod that then references the observable properties of the user using the `ssn:hasProperty` predicate. At this stage, the positioning system knows which observable types of data are available for the authenticated user.

Fetching Observations

Once the positioning system is aware of the types of data that are available, individual observations of these properties can be retrieved. For example, if a *position* property is available, an individual *observation* represents the position at a particular point in time.

With our aim for interoperability of different systems that may not use linked data event streams, the fetching of observations does not distinguish between observations within a stream or observations in a dataset as done earlier in our previous solution [43]. Similar to the fetching of properties, our implementation will attempt to structure new observations according to our proposed solution. However, existing observations or observations made by other services can still be obtained.

```
SELECT DISTINCT ?observation WHERE {
    ?collection tree:member ?observation .
    ?observation a sosa:Observation .
    FILTER(?collection = <...>)
}
```

Listing 4.17: Generic SPARQL query for retrieving all observations in a property collection

To fetch observations, we use a generic SPARQL query as shown in Listing 4.17, executed on the property using link traversal querying [219]. This query retrieves all observations in a property without distinguishing between observations within a stream or dataset. The result is a set of observations that can be used for further analysis and processing.

Creating Properties

A positioning system may provide observational data that is not yet available in a user's Pod. In such cases, the system should enable the creation of new properties and observations. These newly created properties can then be linked to the user's profile card for future reference or other applications that want to make use of this data.

Creating a new property is a three-step process: (1) the property container and `property.ttl` file are created. This `property.ttl` file contains information about the observational property and links to TREE nodes that contain observations about the property. (2) the property is linked from the user's profile card using the `ssn:hasProperty` predicate. (3) the metadata of the property container is set, which includes information about the LDES event stream, its retention policy and other knowledge needed by an application to store new observations of this property.

Creating Observations

When we create a new observation, we first assume that the property exists and that the application can access the container where this property is located. Creating a new observation involves several steps, but it is important to note that it is up to the producer who stores this data to structure and split the data. A producer may split the data depending on the existing amount of data in a single node, the date of the observations within the node or any other relevant criteria. The observations may be appended to the event stream in chronological order, ensuring that the stream maintains its immutable nature.

Our solution makes use of the structuring of LDES, which consists of nodes that each contribute additional data pertaining to a particular collection. In our case, this is a collection of all observations from an observational property.

As an example, a new node can be created whenever the previous node has more than 50 observations or when the last observation in a node was created more than 24 hours ago. In `@openhps/solid`, developers can configure these criteria depending on their use cases. In the current solution, the structuring is based on the implementation of the producer, rather than the user — which is one of the changes that will be made in future work.

Currently, the data producer that creates new observations is responsible for adhering to and managing the data retention policy, which is defined using LDES. In future work, this data retention policy should be managed by an orchestrator or the Solid Pod itself to ensure that data producers are not responsible.

Slabbinck et al. [213] also indicated in their work that maintaining streams within a Solid Pod requires additional logic, which is currently not present in Solid. However, as an architecture for structuring this data, while solving the scalability issue which was present in our initial design, we believe that our proposed solution is a step in the right direction.

4.4.6 Implementation in a Positioning System

In Chapter 3, we already established that we can model a positioning system as a processing graph with individual nodes that either push or pull information. In Section 4.2, we further detailed that we can express these nodes as *procedures* that act upon the data. In this section, we detail how our user-centric storage solution integrates with a positioning system, or more specifically, as the *source* or *sink* of a positioning system.

Sink

A sink node is used to finalise information. It can be a node that visualises a location, saves data to a database, or sends data to another system for further processing. In our context, the sink node can be used to store observational data from the positioning system in the user's Solid Pod. This data can then be analysed, visualised, or shared with other applications that have access to the Pod. The sink node in our solution is responsible for creating new observations and properties in the user's Pod, based on the data received from the positioning system.

Internally, a sink node is relatively in line with the workflow discussed earlier for creating observations. When new data is received from the positioning system, the sink node transforms the data frame into observations. These observations are then stored in the correct LDP container (i.e., TREE node).

Source

A source node, on the other hand, is used to provide information to the positioning system. Other than the sink node, its implementation is more complex due to the active *pushing* of new information from a stream as opposed to passively *pulling* information.

Figure 4.17 illustrates a source node and its role in pushing and pulling information from Solid. Pulling information involves querying the Pod for the latest observation similar to the solution detailed in the previous section. Querying offers a solution

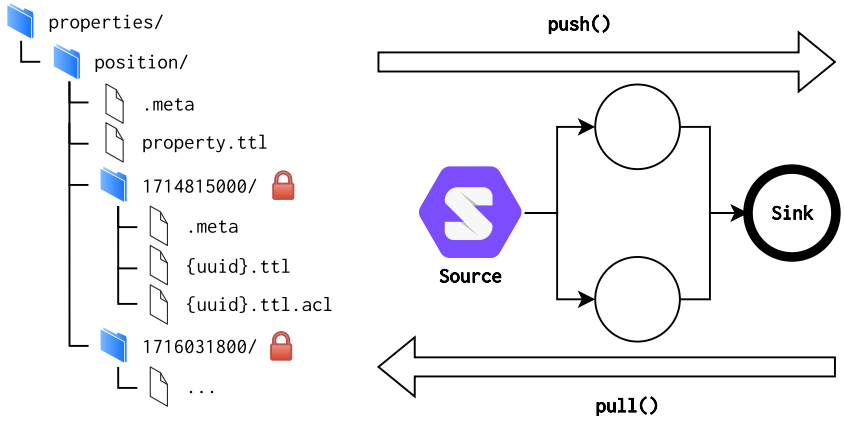


Figure 4.17: Pushing and pulling information from the Solid Pod

to fetch the latest information. However, it does not *force* the creation of new information. Actively pushing new information involves a more complex design, as it requires monitoring the stream of observations from the Solid Pod and reacting to each new observation.

Communication Broker

As detailed earlier in Section 4.4.4, Solid can be used as a communication broker. By simply combining the source and sink nodes in two independent processing graphs, we can push information from one positioning system to the other.

Section 3.6 includes a figure that illustrates the use of MQTT as a communication broker. A similar approach can be taken in our proposed solution, where the sink node stores data in a Solid Pod and the source node actively fetches new observations.

4.4.7 Conclusion

We presented a Solid-based architecture enabling the creation of a user-centric positioning system where user-owned location and sensor data are stored in a user's personal data vault. This user-centric storage allows users to remain in control of their data while allowing different positioning systems and applications to access and use the data, given a user's consent.

We realised a basic implementation of the proposed architecture where two positioning systems and one location consumer application store and read data from the same decentralised vault. This application and implementation are further detailed

in Section 6.4. Based on related work that was published after our initial solution, we modified our final solution to structure the data using Linked Data Event Streams (LDES). We still make use of the SOSA and POSO ontology to represent our data, but we utilise the TREE specification to structure the data over multiple *buckets* that are more scalable than the single resource used in our first prototype.

While Solid is still under development, it highlights the need and advantages of decoupling a user's data from applications that produce and consume the data. With location and sensor data being such a sensitive property of a person, it is highly beneficial to store them in the personal data vaults provided by Solid. We showcased that this change in storage is not only ensuring transparency, but the interoperability of data can help in linking multiple indoor positioning systems and improve the reasoning performed on the data. By using linked data in Solid, we might also leverage the data vaults as communication brokers for transmitting sensor observations from one part of a positioning system to another part by using live update notifications [139].

One limitation in the structuring of data with LDES is the possibility of race conditions that can occur on the event stream. As discussed in our solution in Section 4.4.5, we mentioned that it is up to the data producer to structure the data in a Solid Pod. However, this also entails that multiple producers do not know about the currently running *transactions* that are occurring over the data. This can cause issues such as duplicated nodes, overridden data in shared resources such as the `property.ttl` or in some cases, data that is not linked to the property due to deleted triples. To solve this issue, an orchestrator should be designed that handles the storage and structuring within a Pod. Alternatively, if Solid supported bulk operations where multiple actions can be performed using a single request, our solution could be designed to prevent race conditions.

To further improve interoperability between positioning systems that produce information, future work should use the `ldes:BucketizeStrategy` to determine the appropriate strategy for splitting observations into nodes. Currently, each data producer, including our implementation in OpenHPS, handles these *bucketising* strategies based on the developer's preferences. This strategy can be customised based on various criteria, such as the size of each node, the time interval between nodes, or other relevant factors. While bucketising strategies, retention policies, and data partitioning schemes are common in data streaming applications, applying them to Linked Data streams in a Solid Pod introduces new challenges that need to be addressed. With the potential for multiple producers to be writing to the same Pod simultaneously, conflicts may arise in the structuring of data. Future work should therefore consider the use of an orchestrator that manages the structuring and ensures that the data adheres to the policies and strategies defined in LDES.

The beginning of knowledge is the discovery of something we do not understand.

– Frank Herbert

Chapter 5

Discovering Positioning Systems

In Chapter 4, we defined interoperability of a positioning system as the ability to access, read and understand the data produced by positioning systems. As an answer to RQ2.1, we proposed a user-centric data storage of linked data in a Solid Pod. However, while this enables the seamless exchange of data, it does not allow users or applications to discover the availability of these interoperable positioning systems.

The discovery of a system or service is not necessary when we assume the existence of a single positioning system or standardisation, but it is required when we have decentralisation of systems and services, which is the case for indoor positioning systems. To answer research question RQ3, we delve into the concept of discovering systems and services. The focus of this chapter is on defining a method or framework that enables users to discover the availability of different positioning systems in different environments.

As outlined in Section 2.6.3, from a theoretical point of view, we can use the *open world assumption* of the Semantic Web to query all the necessary information that we could potentially need by traversing the Web. From an engineering point of view, querying the entire Web is not feasible, and we need a more structured approach to finding information, or ideally multiple structured approaches that find the same information.

Knowledge starts with the discovery of something we do not know or understand, as stated by Frank Herbert. The same applies to the interoperability of data. With decentralised data, we do not know of the existence of knowledge. Does the indoor environment you visit have a floor plan? Is there an application available that can help you navigate the building? Are there other services that I, as a user, should be aware of when visiting a particular location? Without knowledge of the existence of positioning systems, the interoperability of such systems is irrelevant.

5.1 Methodology

The discovery of data and services is a broad domain with various solutions depending on the goals. To scope our research on this discovery, we focus on the discovery of indoor positioning systems and the technology used within such systems. An indoor system is deployed in a building on Earth, meaning it has a fixed location.

For discovering these types of services, we investigated methods of discovery that found their foundations amongst standardisation within these services. By doing this, we hope to design a discovery solution that does not break the interoperability of the systems that are being discovered, as outlined with research question RQ3.1. Next, as detailed by research question RQ3.2, since an indoor positioning system is deployed within a certain environment, we also investigated the physical discovery of a positioning system. In general, based on current use cases for indoor positioning systems (see Section 2.7), we can assume that the discovery of such a system is primarily needed when physically present in the environment where the service is used.

In this chapter, we present two contributions that help with the *local* discovery of positioning systems (i.e., physical discovery) but also the *global* discovery, which will aid in the interoperability of our integrated solution of Chapter 7. Our first contribution is a semantic beacon specification called *SemBeacon* that advertises a Bluetooth signal to nearby receivers and describes the environment using the POSO ontology and other ontologies available on the Semantic Web. We decided on the use of Bluetooth beacons due to their prominent use in indoor positioning systems and frameworks.

We validated the use of SemBeacon through various proof of concepts, multiple applications and real-world usage during the IoT 2023 and FOSDEM 2025 conferences to identify and discover users via their WebID. However, since this solution requires additional hardware in a building, we designed the Linked Data Hash Table (LDHT) specification that offers global discovery of positioning systems based on a rough location estimate. Together, SemBeacon and LDHT offer a robust discovery method for positioning systems and the environments in which they are deployed. The related work for our local and global discovery solution is further detailed in Sections 5.2.2 and 5.3.1 respectively.

5.2 Semantic Bluetooth Low Energy Beacons

As an answer to research question RQ3 and the underlying research question RQ3.2, we developed the *SemBeacon* specification [45] to discover physical *things* and

environments. The general principle of our specification is that uniquely identified beacons have a position and additional semantic data available on the Web, which is accessible via a Uniform Resource Identifier (URI), describing the beacon and its deployed environment. Beacons broadcast this URI via an advertisement, allowing any application that receives the advertisement to know the position of the beacon. Because indoor positioning systems often detect multiple of these battery-powered beacons in rapid succession, our specification is designed to limit the number of network connections needed to retrieve information about the location of beacons and the amount of data these beacons have to broadcast. Our specification is fully backwards compatible with AltBeacon and iBeacon [220, 221] and Eddystone-URL scanners, which allow existing buildings to gradually add SemBeacons to cover their spatial area.

With SemBeacon, we aim to provide a solution to discover services and data based on the physical location, but we also want SemBeacons to act as *context providers* in ubiquitous computer systems. Similar to other related work, such as ImogI [79], SemBeacons can support location-aware adaptivity, providing more detailed floor plans when nearing an area. With ImogI, the authors envisioned that a museum would use intelligent artefacts which broadcast their information. Similar to ImogI, the design philosophy with SemBeacon is to avoid the need to update a centralised database by decentralising the data on the Semantic Web.

While the Eddystone-URL specification can be used to broadcast a URI of a semantic resource providing more information about an object, there is a lack of information for its use in real-time positioning systems. An Eddystone beacon's MAC address can be used to uniquely identify an object. However, as the URL needs to be a short encoded URL to fit within the 17 bytes, the information that can be encoded within this URL is limited. Unlike regular-sized URLs, where there is enough space to support descriptive domains and paths that provide context about the content (e.g., `mycompany.com/building1`), 17 bytes is too limited to recognise beacons belonging to the same spatial area. Furthermore, the use of URL shorteners is encouraged with Eddystone-URL, which further obfuscates the URL. Due to this limitation, beacons within the same spatial area cannot be easily recognised based on this short URL.

Being able to identify a set of beacons belonging to the same spatial area or *namespace* prevents us from having to perform network requests to retrieve this information. One possible solution to this problem is the use of Eddystone-UID, which offers a UID and namespace to classify the hierarchical structure of a beacon. An Eddystone-URL could be combined with this UID frame as an advertisement packet and scan response to offer similar capabilities as SemBeacon. However, due to their packet size, it is not possible to encode additional information about the type of SemBeacon that is being deployed as shown in Section 5.2.3.

Other than the local discovery of LOCgram [26], which also uses Eddystone-URL beacons for discovering nearby location-based services, we want to offer a method that distinguishes between regular URLs and URIs that provide semantic data. In addition, our focus with SemBeacon is towards positioning systems by adding additional flags and identifiers that can help with the identification of useful beacons.

With our solution, we also aim to create interoperable indoor positioning systems where users can obtain the location of a beacon as well as any other sensors or configured positioning systems in a building without prior knowledge of the environment. Our proposed solution for creating semantic Bluetooth Low Energy (BLE) beacons is divided into two main sections. While Section 5.2.3 describes the advertisement specification based on AltBeacon and Eddystone-URL, Section 5.2.4 introduces the semantic description of SemBeacons on the Web, including the use of the POSO [40] vocabulary for describing indoor positioning systems.

A main advantage of using SemBeacon for broadcasting information about an environment is the ability to operate without the need for a proprietary application that defines a database or Web Service to map the beacons to contextual data. Our vocabulary enables the description of positioning systems which in turn helps applications to find these beacons in an indoor environment where GPS cannot be used.

5.2.1 Architecture

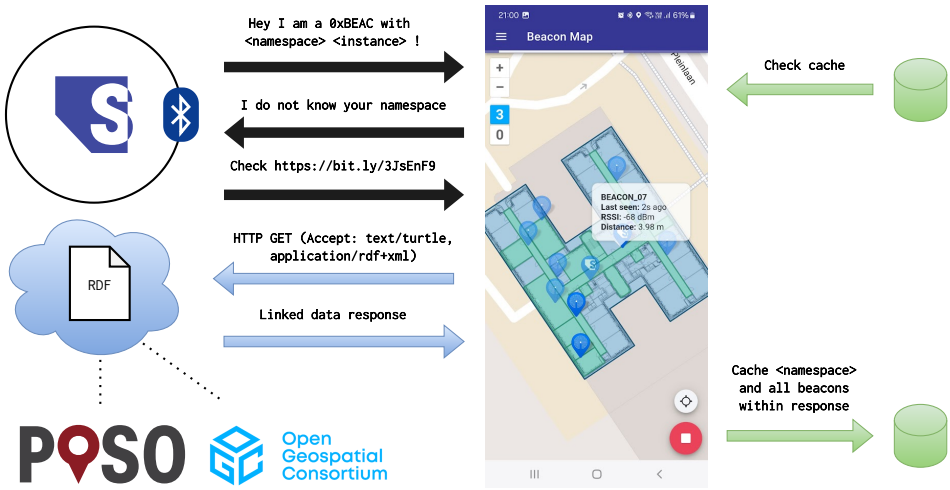


Figure 5.1: SemBeacon architecture and discovery flow

Figure 5.1 illustrates the SemBeacon architecture and discovery flow. The beacon acts as a *lighthouse* that emits a namespace and instance identifier that can be used to uniquely identify the beacon. An application checks if it has prior knowledge

about the namespace. If yes, it means that the beacon or another beacon in that same namespace was previously discovered. If no information is available, the application will request additional information from the beacon, after which it will respond with a URI.

This URI leads to contextual information about the specific beacon. Depending on the use case, it can contain information about its position, observable data such as telemetry information, and a description of how to interact with the beacon. However, more importantly, it also links to the namespace in which it is deployed and all other beacons, infrastructure or environment information related to this namespace. The information is represented as linked data and uses vocabularies such as the POSO ontology and GeoSPARQL.

After the beacon's information is fetched, the namespace, along with other information on devices and beacons within this namespace is cached. When new beacons are detected with the same namespace, the information does not need to be retrieved. Furthermore, the contextual information of non-SemBeacons, such as iBeacon can be distributed by SemBeacons, preventing the need to replace existing infrastructure with SemBeacon.

5.2.2 Related Specifications

Our SemBeacon solution draws its inspiration from existing specifications, primarily from iBeacon, AltBeacon, Eddystone-URL, UriBeacon and Bluetooth IPS. In this section, we delve deeper into these specifications and their uses within an indoor positioning system.

iBeacon Specification

The iBeacon specification was developed by Apple Ltd. in 2013. In the specification, a total of 30 bytes is used for the *manufacturer data* to encapsulate the information it broadcasts. As illustrated in Figure 5.2, the iBeacon specification offers three layers of device identification (dark blue) and adds an additional byte for the reference transmission power at one metre distance. A 128-bit *proximity UUID* identifies all beacons used by the same positioning system or application. Next, an unsigned 16-bit *major* and *minor* number identify the hierarchy of the beacon with a spatial region [222]. The specification does not mention at what spatial level the hierarchical separation of the major and minor identifiers should be chosen. However, most commonly, the major number defines the building or floor and the minor number represents the beacon within that floor. The iBeacon specification does not advertise a beacon's location and requires a database or a known trusted service to provide this information.

Adv Flags 3B	Len 1B	Type 1B	Company ID 2B	Beacon Type 1B	Beacon Len 1B	Proximity UUID 16B	Major 2B	Minor 2B	TX @ 1m 1B	
-	0x1A	0xFF	0x4C00	0x02	0x15	*uint8[]*	*uint8*	*uint8*	*int8*	

Figure 5.2: iBeacon advertisement data (30 bytes)

Eddystone Specification

Eddystone was developed by Google in 2015. Other than the iBeacon specification, it does not use custom manufacturer data, but creates a service with its own unique UUID containing the beacon information [223]. The specification offers four types of *frames*:

- **Eddystone-URL** for broadcasting short URL addresses. Originally, this frame was used in the Physical Web to show notifications on Android devices.
- **Eddystone-UID** for broadcasting beacon identifiers similar to iBeacon. Uses a namespace and instance identifier instead of a service UUID, major and minor number.
- **Eddystone-TLM** for broadcasting telemetry data such as sensor data or battery information.
- **Eddystone-EID** is a security specification for broadcasting ephemeral identifiers. In comparison to Eddystone-UID, the identifier changes at a given interval and has to be resolved via an external service. This ensures that the beacon cannot be tracked based on this rotating identifier.

Figure 5.3 shows the Eddystone specification for UID and URL Eddystone frames. To limit the size of the broadcasted URL in an *Eddystone-URL*, the specification adds 1 byte to specify the URL scheme prefix (e.g., `0x00` matches `"http://www."` or `0x01` matches `"https://www."`). The URL itself is a UTF-8 character array of the URL without a scheme. Commonly used UTF-8 character groups, such as top-level domains, can be encoded using a single byte (e.g., `0x00` matches `".com/"`) as detailed in the specification[1].

Despite the encoding techniques, a URL shortening service is often needed in order to obtain a URL that fits within the frame. This short URL redirects to the full URL, preventing the encoding of identifiable information within the domain or path without performing an HTTP request.

Multiple frames can be combined to provide additional data. For example, the Eddystone-UID frame can be combined with a scan response containing telemetry data to return additional data while identifying a beacon with its namespace and instance identifiers.

[1] `https://github.com/google/eddystone`

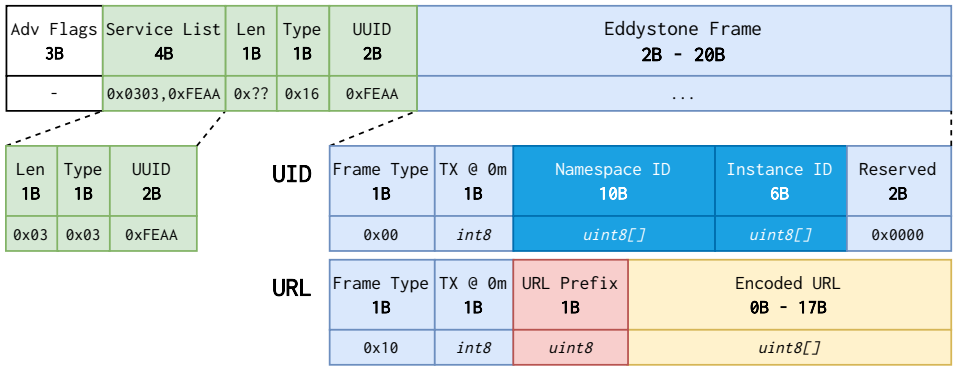


Figure 5.3: Eddystone-UID and Eddystone-URL advertisement data (13–31 bytes)

Using the Eddystone-URL frame, Seo and Yoo [224] proposed an interoperable context model advertising *Place*, *Object* and *Annotation* URIs. These URIs led to HTML web pages providing augmented reality context via HTML elements. For its use within the context of interoperable positioning systems, there is a lack of identifiable information that would prevent applications from having to access the URI of every encountered Eddystone-URL beacon.

UriBeacon Specification

As the predecessor of the Eddystone-URL beacon frame [153], UriBeacon has a similar maximum URI byte size as Eddystone (17 bytes) and uses the same encoding techniques. The main difference to the Eddystone-URL specification is the addition of 1 byte that can be used to add flags to the URI. The specification only implemented one *invisible hint* flag defining whether users should be notified about the presence of the URI. In Eddystone-URL, this byte is used to indicate the type of Eddystone frame (i.e., `0x10` for Eddystone-URL). Having the ability to add flags to the beacon to indicate how the client should handle or use the URI is an important and useful feature in real-time systems, which we also decided to use in our own solution. Figure 5.4 showcases the UriBeacon specification. As can be seen from the figure, there are no differences between Eddystone-URL and UriBeacon apart from the service UUID and flag byte.

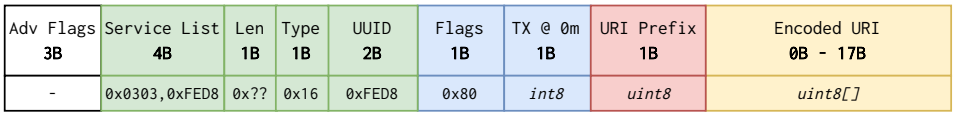


Figure 5.4: UriBeacon specification

AltBeacon Specification

AltBeacon is an open specification by Radius Networks[2] that is backwards compatible with iBeacon scanners not specifically scanning for beacons manufactured by Apple Ltd. However, unlike iBeacon's identification through a proximity service UUID and a major and minor classification, AltBeacon uses 20 bytes as a single beacon identifier.

Adv Flags 3B	Len 1B	Type 1B	Company ID 2B	Beacon Code 2B	Beacon ID 20B	TX @ 1m 1B	Unused 1B
-	0x1B	0xFF	*uint16*	0xBEAC	*uint8[]*	*int8*	-

Figure 5.5: AltBeacon advertisement data (31 bytes)

While iBeacon only uses 30 bytes, the last byte in AltBeacon was added to the specification as a manufacturer-specific byte. The specification recommends using a UUID for the beacon identifier's first 16 bytes in a single *organisational unit*, enabling backwards compatibility with iBeacon scanners looking for a proximity UUID.

An AltBeacon is identified as such by setting bytes 7–8 to the `0xBEAC` hexadecimal representation, which would indicate the beacon type and remaining payload length for iBeacon. In AltBeacon, these two bytes are called the *beacon code* as illustrated in Figure 5.5. Similar to iBeacon, the AltBeacon specification does not include information about a beacon's location. Unlike the AltBeacon specification, which recommends using the same 128-bit UUID for the same organisational unit, we aim for a specification where beacons are not specifically deployed for a single proprietary application.

All the presented prominent beacon specifications used for positioning and proximity awareness are built on top of Bluetooth v4.2 compatible advertisements. While these advertisements are lightweight and sufficient to broadcast the required information, they are limited when used without an additional database or Web service mapping the beacons to other contextual data. With our SemBeacon specification, we have considered the backwards compatibility with the prominent beacon specifications, as well as some of their characteristics and features that could be useful for semantically describing the locations and environments in which these beacons are placed.

Bluetooth IPS Specification

In 2015, the Bluetooth SIG published a core specification for advertising an indoor positioning service via BLE [225]. This advertisement specification allows broadcasting the global WGS84 location [65], local coordinates within a building, the

[2] `https://github.com/AltBeacon/spec`

transmission power and additional information needed to know the transmitter's location.

Figure 5.6 shows an overview of the Bluetooth IPS specification as described by the Bluetooth SIG. The specification describes the broadcast of the location data in 20 bytes and GATT services to configure these properties. Any changes made to the location have to be configured in the hardware itself, making remote changes impossible without additional network communication. While the specification broadcasts a location in a global and local reference frame, no information is broadcast on how to interpret these local coordinates.

Adv Flags 3B	Len 1B	Type 1B	Flags 1B	Lat 4B	Long 4B	North 2B	East 2B	TX Power 1B	Floor 1B	Altitude 2B	Uncertainty 1B	RFU 1B
0x020106	0x14	0x25	-	-	-	-	-	*int8*	-	-	-	-

Figure 5.6: Bluetooth IPS specification (20 bytes)

Other than all aforementioned specifications that influenced the design of our solution, Bluetooth IPS provides an interesting advantage that other specifications do not. The advertising payload itself provides basic context about the location instead of relying on external information.

5.2.3 Advertisement Specification

For our semantic beacon specification, we had several design requirements. The specification should be compatible with both the iBeacon as well as AltBeacon specifications to be recognised by existing deployed indoor positioning systems. In addition, existing infrastructures consisting of hardware beacons whose protocol cannot be altered should also be supported, regardless of the type or manufacturer of those beacons. Our advertising packet payload shown in Figure 5.7 is based on the AltBeacon specification which is backwards compatible with iBeacon.

We also introduced a BLE v5 compatible advertisement leveraging the additional payload size and range of BLE v5 [91]. Figure 5.8 showcases the extended advertisement data with the same 16-byte namespace and 4-byte instance identifier. However, we added a byte for versioning the specification and an extended encoded resource URI of 128 bytes, to avoid the shortening of URLs.

Identification

A SemBeacon is recognised as an AltBeacon (i.e., bytes 7–8 should be `0xBEAC`) that offers an Eddystone-URL compatible scan response. Individual SemBeacons are identified as a combination of a 16-byte *namespace identifier* and a 4-byte *instance identifier* that is unique for all beacons in the same namespace. The namespace is a 128-bit universally unique identifier (UUID) that is unique per spatial area where

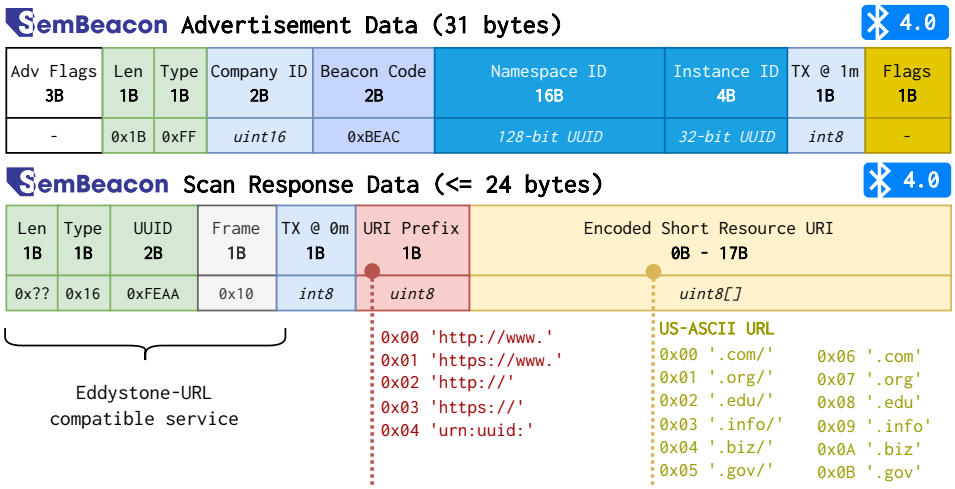


Figure 5.7: SemBeacon specification based on AltBeacon with an Eddystone-URL compatible scan response

beacons are deployed, including buildings or even floors. The instance identifier is a 32-bit (unsigned big-endian integer) uniquely identifying all beacons within the same namespace. The splitting of an identifier into a namespace and instance is inspired by Eddystone-UID. However, unlike Eddystone-UID, which recommends that the namespace be used for filtering beacons during the scanning (i.e., for a specific application), we utilise the namespace to remember which semantic resources have already been retrieved.

Similar to the iBeacon specification which does not specify the spatial relation between its major and minor identifiers, SemBeacon does not force the use of a certain spatial hierarchy between namespaces and instance identifiers. However, all beacons that can be fetched with a single HTTP GET request should be considered

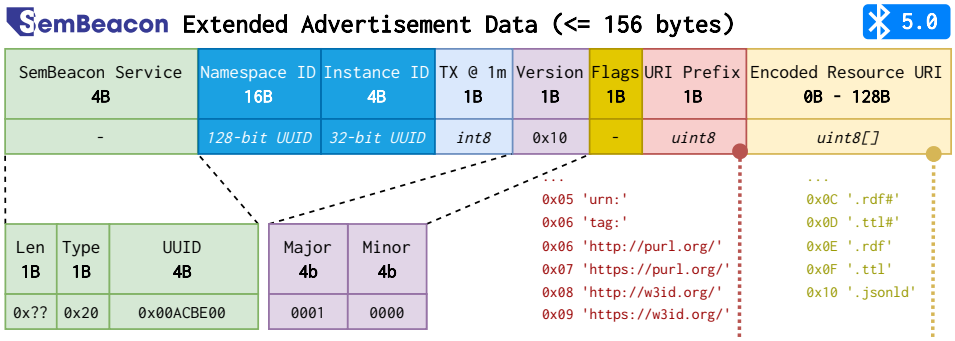


Figure 5.8: SemBeacon specification for Bluetooth 5.0

to be in one *namespace*, offering users the freedom to decide how to group beacons in a single resource.

With our aim for interoperability, newly developed BLE scanners should not filter on a specific namespace identifier, but rather perform filtering after obtaining the semantic data. However, existing implementations that scan for certain *proximity UUID* iBeacons can be supported by using the same 16-byte UUID for the namespace identifier. These existing implementations will not be able to leverage the broadcasted URI, but can still use an internal database — enabling existing positioning systems to slowly transition to SemBeacon without having to modify the existing application. Applications that want to implement filtering of SemBeacons used by one positioning system can use a prefix for the 128-bit UUID.

Semantic Flags

With SemBeacons, we aim to provide most, if not all, information on the Semantic Web [124] via the broadcasted URI. However, some beacons might not require an HTTP request to the URI if the online information is guaranteed to be unusable for the application detecting the beacons. For example, an application designed to use SemBeacons as landmarks for indoor positioning cannot make use of beacons attached to movable objects.

The final byte of the SemBeacon advertisement is a manufacturer-specific byte in AltBeacon. SemBeacon uses this byte to provide boolean information about the type of SemBeacon and the expected available online data, which can be used to decide whether to access the resource URI depending on the scanning application.

Bit	Description
0	Indicates if beacon has a position (0 = unsure, 1 = yes).
1	Indicates if beacon is private (0 = public, 1 = private).
2	Indicates if beacon is stationary (0 = stationary, 1 = mobile).
3	Indicates if the beacon is part of a positioning system (0 = no, 1 = yes).
4	Indicates if beacon provides observable data (0 = no, 1 = yes).
5	Indicates if beacon provides actuators (0 = no, 1 = yes).
6–7	Reserved for future use.

Table 5.1: SemBeacon flags

In Table 5.1 we provide an overview of the available SemBeacon flags, which are based on the Bluetooth Indoor Positioning Service [225] flags, Eddystone frames and the UriBeacon [153] flag. While they repeat certain information that could be available online, the flags depend on the deployment of the beacon and should not need to change after deployment. For example, if a beacon is placed on a wall and is therefore considered stationary with a fixed position, these properties of the beacon should not change regardless of any external influence. Our current version of

the specification (version 1.1) introduces some changes to the originally published specification. Our previous version of these flags can be found in Appendix C.1.

To support positioning through multilateration or cell identification as outlined in Section 2.3, each beacon requires a position. During the installation of beacons in the physical environment, the first bit is set to 1 to indicate that the beacon is installed at a fixed known location. In a use case where these beacons are placed to merely provide information about the environment or provide information about other beacons in the environment, they do not necessarily require a location. For example, a building owner may place a SemBeacon at the entrance of a building to make nearby users aware of the other beacons available within the building, the environment, positioning system(s), and other contextual data.

If users or assets are equipped with a SemBeacon, these moving objects (indicated by bit 2) can be tracked. For some assets or persons, the semantic description might require authentication to limit access to the URI. In this case, the private flag at bit 1 can be toggled on. A use case would be that the semantic data is stored in a Solid *Pod* [139] as discussed in Section 2.5.2. In such a scenario, the SemBeacon advertises a privately owned resource inside a Pod. In Section 6.7, we demonstrate a use case where SemBeacon is used to advertise a privately owned augmented reality environment, managed by a Solid Pod.

Finally, interoperable indoor positioning applications might only be interested in SemBeacons which are placed for indoor positioning systems, in which case bit 3 can be toggled to indicate that the beacon forms part of a positioning system. In addition, the semantic description might provide additional information on implemented positioning techniques that use other sensors such as Wi-Fi fingerprinting or Ultra-wideband beacons [226, 54, 2].

Bit	**Axiom**
0	`poso:hasPosition` **min** 1 `poso:AbsolutePosition`
1–2	N.a.
3	`poso:inDeployment` **min** 1 `ssn:Deployment`
4	`ssn:hasProperty` **min** 1 `sosa:ObservableProperty`
5	`ssn:hasProperty` **min** 1 `sosa:ActuableProperty`
6–7	Reserved for future use.

Table 5.2: SemBeacon flag axioms

With flags being used to determine if information is available online, the use of these flags implies that certain data is indeed available online. Table 5.2 lists the axioms for the 6 available flags. As an example, if a SemBeacon indicates to provide observable data, this data should be available through the `sosa:ObservableProperty` predicate. Similarly, other flags should imply that predicates are available in the online linked data.

Scan Response Payload

Bluetooth Low Energy can respond with a scan response payload in addition to the advertisement data. The scan response consists of an additional 31 bytes of data that is sent whenever a scan request is received. In Figure 5.7, we see an Eddystone-URL-compatible SemBeacon URI in the scan response. We have chosen this frame as a legacy scan response due to its implementation in existing libraries, applications and hardware. During the setup of the beacons, the URI can be put behind a linked data front-end that serves a web page when the URI is visited by the browser of users who are scanning for Eddystone-URL beacons. These websites might contain a visual representation of the semantic data as in the Physical Web [152]. However, unlike the Physical Web, our main goal is to access the URI via software. A SemBeacon URI should resolve to a machine-readable document where the data is structured using RDF. The namespace identifier should match the base URI, ensuring that the namespace identifier matches the base URI for all beacons within this document.

5.2.4 Semantic Description

For the design of our vocabulary to semantically describe beacons, we have focused on the use case where SemBeacons are used as proximity beacons in positioning systems and location-aware applications, but additional functionality and data can be added as with any vocabulary. We have chosen the Positioning System Ontology (POSO) [40] as our core ontology. The ontology already offers the terminology to describe *landmarks* used in a positioning system, such as beacons or other RF transmitters. We extended POSO (`poso-common`) with an additional vocabulary for describing the different types of beacons including SemBeacons and their related information.

As outlined in Section 5.2.3, the identification of a semantic beacon is done on a *namespace* and *instance* level. Namespace identifiers resemble the base URI of a semantic resource. In our online technical specification, we make it clear that multiple beacons within the same *resource* should use the same namespace. A resource is a collection of triples that can be obtained from a single HTTP GET request.

In Listing 5.1, we show a basic description of a SemBeacon and its spatial location written as triples using the Resource Description Framework (RDF) [227]. The individual `:room_a1_2` on line 13 is a `sembeacon:SemBeacon` type. An individual is an instance of a class, in this case for a specific *Deployment* [124]. We describe the reference RSSI at 1 metre and the absolute position defined using the POSO ontology. As some positioning systems and beacons require the reference RSSI to be measured at 0 metres (e.g., Eddystone-UID), the distance can be identified using `poso:hasRelativeDistance` on line 22. Further, we use

http://sembeacon.org/example.ttl

```
1  @prefix :          <http://sembeacon.org/example.ttl#> .
2  @prefix hardware:  <http://w3id.org/devops-infra/hardware#> .
3  @prefix poso:      <http://purl.org/poso/> .
4  @prefix posoc:     <http://purl.org/poso/common/> .
5  @prefix sembeacon: <http://purl.org/sembeacon/> .
6  @prefix qudt:      <http://qudt.org/schema/qudt/> .
7  @prefix unit:      <http://qudt.org/vocab/unit/> .
8
9  :building_a a ssn:Deployment ;
10   rdfs:label "Building A" ;
11   sembeacon:namespaceId "e19c5e1ed6a14d..."^^xsd:hexBinary .
12
13 :room_a1_2 a sembeacon:SemBeacon ;
14   rdfs:label "SemBeacon Room A1.2"@en ;
15   rdfs:isDefinedBy <http://sembeacon.org/example.ttl#> ;
16   sembeacon:namespace :building_a ;
17   sembeacon:instanceId "beac0101"^^xsd:hexBinary ;
18   hardware:mac "00:11:22:33:44:55" ;
19   posoc:referenceRSSI [ # Reference RSSI is a ...
20     poso:hasRSS [ # ... factory calibrated signal strength
21         qudt:unit unit:DeciB_M ; qudt:numericValue -56 ] ;
22     poso:hasRelativeDistance [  # ... measured at a distance
23         unit:Meter ; qudt:value "1.0"^^xsd:double ]] ;
24   poso:hasPosition [ a poso:AbsolutePosition ;
25     poso:hasAccuracy [ ... ] ; poso:xAxisValue [ ... ] ;
26     poso:yAxisValue [ ... ] ; poso:zAxisValue [ ... ] ] .
```

Listing 5.1: Example SemBeacon description

the Hardware Ontology from the DevOps Infrastructure Ontology Catalogue [228] to provide a beacon's physical address, which can be used by some positioning systems to uniquely identify other types of beacons that do not offer an identifier in their advertisement data.

Listing 5.2 illustrates how additional beacons defined in the same resource and namespace can be retrieved whenever the resource is fetched from SemBeacons. An iBeacon is defined with a deployment as a common namespace also shown in Listing 5.1 on line 16. In the online documentation[3], we provide more details on all additions to the vocabulary and its usage. Developers can decide to extend the vocabulary to describe different output data types or even different protocols for interfacing with the beacon or device.

[3] https://sembeacon.org/terms/1.0/

http://sembeacon.org/example.ttl

```
:room_a1_3 a posoc:iBeacon ;
  rdfs:label "iBeacon Room A1.3"@en ;
  rdfs:isDefinedBy <http://sembeacon.org/example.ttl#> ;
  sembeacon:namespace :building_a ;
  hardware:mac "00:55:44:33:22:11" ;
  posoc:major 10001 ; posoc:minor 12831 ;
  posoc:referenceRSSI [ ... ] ; poso:hasPosition [ ... ] .
```

Listing 5.2: Example iBeacon description

Figure 5.9 illustrates the extension to the POSO ontology. The POSO ontology already provides the concept of `BluetoothBeacon` as *landmarks*. The POSO-common extension provides common beacon types, such as iBeacon, AltBeacon and an Eddystone beacon. With our SemBeacon extension, we provide an additional SemBeacon class which provides an instance identifier, version and short resource URI data type. The extension also adds a namespace identifier data property and a namespace object property, which can be used on any beacon type. These can be used to link all beacons belonging to the same namespace. The full ontology extension is available in Appendix C.4.

5.2.5 State and Discovery Flow

We now provide an overview of the different steps an application has to take to scan and use SemBeacons. All beacons within the same spatial environment will be put in the same resource. Each beacon within this resource will have the same namespace identifier, which removes the need for unnecessary HTTP requests when multiple beacons are detected with the same namespace.

In the following, we list the six different steps of an application scanning for BLE v4 advertisements:

1. **Passive scanning**: The application passively scans for incoming advertisements until AltBeacon-compatible manufacturer data is detected that includes a 128-bit UUID (i.e., namespace identifier). When a UUID is detected, the application will proceed to the beacon identification step.

2. **Beacon identification**: A beacon is identified with a namespace and instance identifier. The application checks if it knows the namespace identifier. If the namespace was not previously discovered, an active scan is performed.

3. **Active scanning**: The application actively scans for new beacons using a BLE scan request. This scan triggers a response from nearby beacons that receive this scan request.

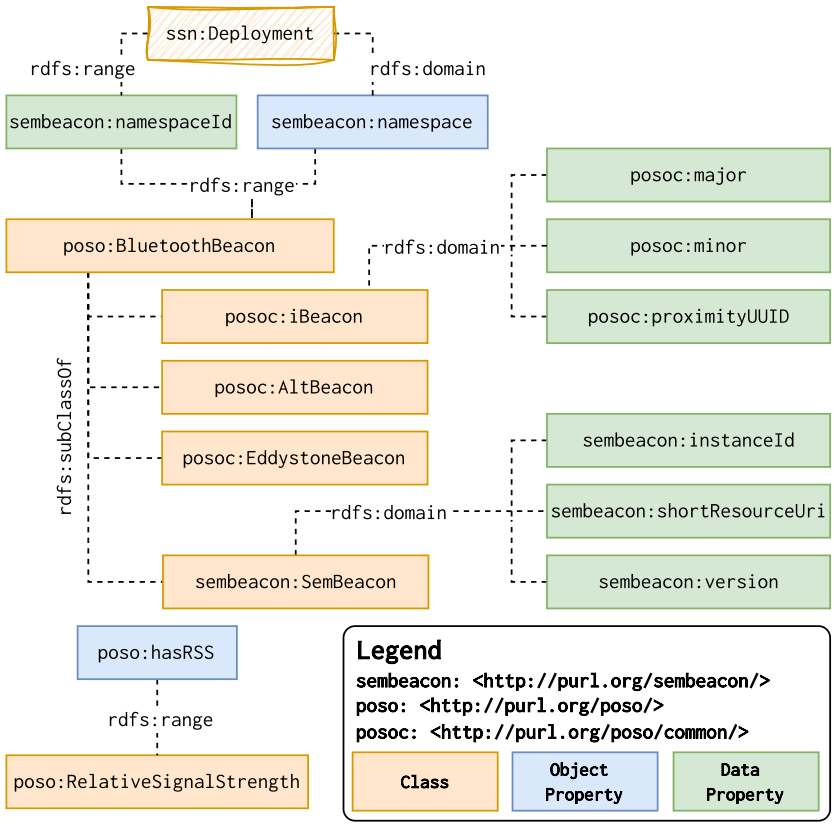


Figure 5.9: SemBeacon extension to the POSO ontology

4. **SemBeacon detection**: SemBeacons will respond with a scan response that includes the Eddystone-URL-compatible resource URI. Compatible scanners will detect a SemBeacon when it has an AltBeacon advertisement and a scan response that is compatible with Eddystone-URL.

5. **Data retrieval**: In case no information about the namespace was available, the resource URI is accessed to retrieve the location of the beacon and other beacons within the same namespace. Depending on the flags shown earlier in Table 5.1 and the implementation of the application to act on these flags, the data might not be retrieved.

6. **Passive scanning**: The application continues the passive scan until an unknown namespace is found, in which case step (3) is performed again.

On iOS devices, Apple limits the passive scanning for iBeacon manufacturer data to beacons with a known proximity UUID[4]. This behaviour works fine when you are tracking a set of known iBeacons, but in our use case, this would require prior knowledge of the namespace identifiers of each SemBeacon. However, with our SemBeacon specification, we can circumvent this limitation by scanning for the `0xFEAA` service, which will provide us with a URI. Using this URI, we can then retrieve information about the beacon's namespace — which in turn will provide us with a UUID that we can use to scan for a set of beacons.

5.2.6 Service Ranking

In the service ranking step, once the application has successfully scanned and retrieved data from SemBeacons, it can rank the services based on proximity and relevance to the application. This method of ranking is a discovery technique similar to contextual discovery [154, 155].

5.2.7 Libraries and Application

We implemented and validated the SemBeacon solution by developing our own hardware and software libraries to create and detect these beacons. Using this software, we developed several use cases where SemBeacons are used to discover resources on the Web and in Solid Pods. As a demonstrator of discovering environments and semantic beacons located within this environment, we transformed and redeployed the indoor positioning system used for our OpenHPS framework. When redeploying this positioning system, we made sure to add several SemBeacons near the entrance to broadcast information about the environment [35].

Hardware

Our experiments were performed using custom beacons that we created to aid our research. The beacons were designed based on the ESP32-S3[5], which at the time when we started developing the hardware was a novel new chip that supported Bluetooth Low Energy 5.0 and was initially not available in development boards.

Figure 5.10 showcases the hardware consisting of *(a)* a USB-A port to easily power the beacon using a power bank or existing USB port of monitors, *(b)* an external SMA antenna mount and *(c)* the ESP-S3 together with 16MB of flash. We envision that SemBeacons are deployed to provide contextual information about other beacons or positioning hardware within the building, rather than being used themselves as a positioning technique (i.e., RF-based positioning using Bluetooth beacons). Therefore, the inclusion of the USB-A port facilitates the deployment

[4] https://stackoverflow.com/questions/20988671/how-to-scan-ibeacon-signals-without-specifying-uuid

[5] https://www.espressif.com/en/products/socs/esp32-s3

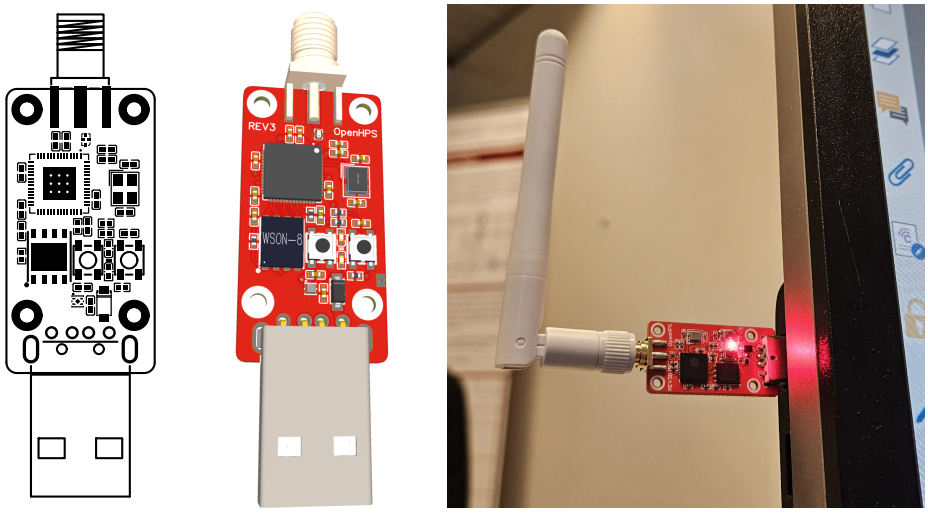


Figure 5.10: SemBeacon with Wi-Fi capabilities and BLE 4.0 and 5.0

in the building by simply inserting these devices into existing hardware such as monitors, routers or other devices.

As part of our goal to ensure easy adoption of SemBeacon, an Arduino library that enables easy development of SemBeacons on any ESP32 development board was open-sourced alongside the hardware[6]. A code snippet with an example of how to use this library, as well as more information about the hardware can be found in Appendix C.

Mobile Application

We have developed a mobile application and a web app for detecting SemBeacons and retrieving the information that these SemBeacons contain. Our app was built for Android, iOS and the Web using the Web Bluetooth API [183].

For our integrated solution, we have expanded on this mobile application to include Solid [139] support to advertise WebIDs to enable the discoverability of users and organisations. In addition, the Solid authentication enables access to URIs that are not public and only accessible by authenticated users.

Figure 5.11 shows several screenshots of the mobile application. The screenshot at the bottom left shows an indoor environment fetched from SemBeacons. This indoor environment includes GeoJSON data to represent rooms, hallways and the building contour, as well as a floor plan image of the floor. On the left of this screenshot, we can see the different floors that are available within this

[6] https://www.arduino.cc/reference/en/libraries/esp32_sembeacon/

environment. On top of this map, we display the beacons as markers. Beacons fetched from SemBeacons, but are not detected through Bluetooth, are shown with a translucent marker to indicate that they are not detected. Users can click on the markers to see more contextual information, such as the distance and RSSI value, as well as the name of the beacon.

In the screenshot on the top left, we have a basic overview of all detected beacons. Our application scans for the most common types of beacons, including iBeacon, AltBeacon, all Eddystone frames and of course SemBeacon. If a SemBeacon can provide contextual information about other non-SemBeacons, such as their name, it will be displayed in this overview. By changing the tab to *Simulator*, the user can use the smartphone to simulate a SemBeacon. The top right screenshot shows an overview of the details of a SemBeacon that is being simulated, including the URI (long and shortened), as well as the flags. Finally, in the bottom right screenshots, we illustrate a simulated beacon for the WebID of the user, which is a SemBeacon with more contextual information to indicate that it is a profile that is being broadcast.

5.2.8 Conclusion

In this section, we presented our solution to the local discovery of positioning systems and digital twins of environments. Our local discovery uses the properties of RF signals to only propagate near a transmitter to ensure that only nearby receivers can obtain information on the environment. By developing the SemBeacon hardware and corresponding Arduino library, as well as the mobile application and web app for detecting and retrieving SemBeacon information, we have provided a comprehensive set of tools for enabling the use of SemBeacon in existing and future applications.

One of the disadvantages of SemBeacon is its requirement for proprietary hardware that can scan for Bluetooth Low Energy advertisements and scan responses. However, our solution expands on existing specifications such as AltBeacon, Eddystone-URL and the Semantic Web to enable the discovery. Furthermore, we also provide an alternative global discovery method in Section 5.3, which provides a discovery method that only requires link traversal querying.

In future work, we plan to extend the semantic flags to add more offline contextual information to the beacons. These flags will depend on SemBeacon's use cases and will not influence the current implementations. We envision SemBeacon being used for future user interactions with IoT devices. These devices could store a stream of sensor data in an online storage provider, which SemBeacon could then advertise to nearby applications or other IoT devices.

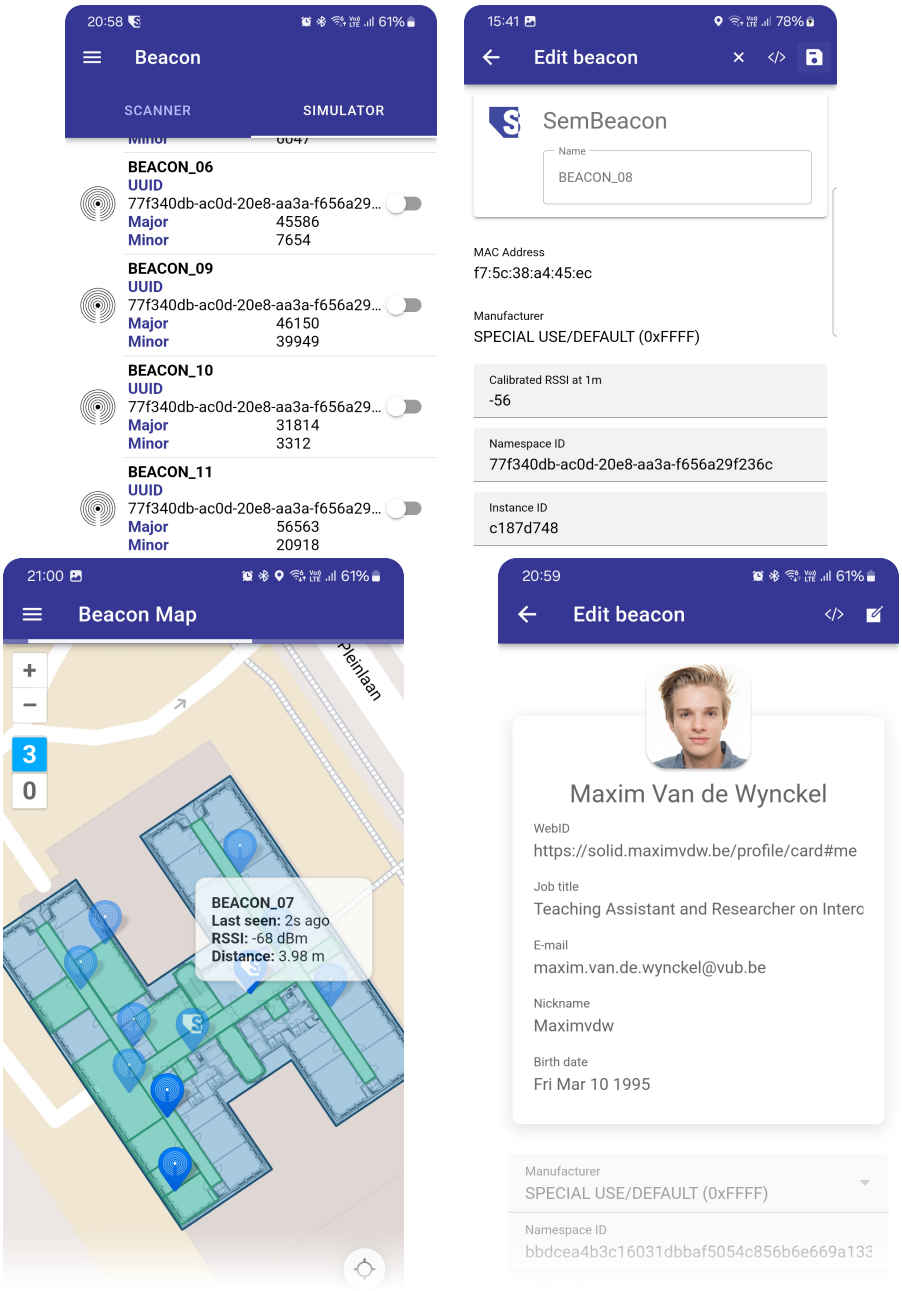


Figure 5.11: Screenshots of the SemBeacon mobile application showcasing the environment and beacon map, WebID broadcasting and other functionalities

5.3 Linked Data Hash Table

Semantic beacons enable local discovery of positioning systems by advertising their presence to local Bluetooth receivers. To aid with the global discovery, as well as the discovery of positioning systems that do not make use of SemBeacons, we designed a distributed lookup table for positioning systems and location-based services. Such a solution forces positioning systems to publicly register to a distributed system, but it avoids deploying hardware in a building.

The lookup table in our solution relies on a geospatial lookup. The user or tracking device provides a rough estimate of a location, based on GPS, cellular signals, an IP address or even user input. In the lookup table, this information is used to determine nearby positioning systems or location-based services that could potentially cover the area the user is currently in. Each of these systems can provide additional context that can help to decide which system is the most appropriate to use for a given application.

One of the main challenges with this approach is the granularity of the geospatial lookup. What3Words [229] is a reverse geocoding service that splits up the earth into a grid that assigns three words to uniquely identify each cell in this grid. Each cell covers three square metres, making it a relatively precise method to describe a location with only three words.

However, in our approach, we want a system to look at which services potentially lie within a larger area, rather than a specific three-square-metre cell. This requires a more flexible and scalable geospatial lookup mechanism that can handle varying levels of granularity based on the accuracy of the rough location that can be obtained. Our proposed solution should be able to locate or discover potential positioning systems. However, access to these positioning systems can require additional authentication.

To solve this issue, we propose the use of a Distributed Hash Table (DHT) consisting of multiple nodes to store and retrieve information. Each node in the DHT is responsible for a set of keys and is able to efficiently retrieve information about the values (i.e., URIs). Our DHT is based on Kademlia [230] with a similar lookup strategy.

To keep this lookup table interoperable, we make use of linked data to store the information of individual nodes and locate nodes using their URI. We created the Linked Data Hash Table (LDHT) specification, which builds on top of the TREE hypermedia specification [216]. The linked data DHT network is implemented in the `@openhps/dht`[7] module of OpenHPS and interfaces with Solid, as shown in Section 2.5.2, to manage each node in the network.

[7] `https://openhps.org/docs/dht/`

LDHT is a solution to a wide range of discovery problems and not only for discovering positioning systems. In our use case, we want to use the specification to distribute the knowledge of indoor positioning systems bound to spatial locations. We apply a hash function to partition the Earth into a fixed-size grid. Each cell in this grid is assigned a unique key that maps to a set of URIs of indoor positioning systems within this cell. Users obtain an inaccurate location via cell towers, GPS or user input. This location is then hashed to the cell(s) that lie within this location radius, after which the LDHT can be traversed to find a list of potential IPSs. LDHT provides a solution to the general research question RQ3 while maintaining interoperability (RQ3.1) by fragmenting a single *collection* over multiple linked data resources. We envision a shared collection of *positioning systems* that is expanded using LDHT nodes that contain part of the ever-growing collection of (indoor) positioning systems.

5.3.1 Related Work

Global discovery is a common challenge to overcome in discovering data and services. In Section 2.6.1, we already discussed several protocols and specifications to perform global discovery of data and services. However, often, these approaches require centralised registries, which are not scalable.

The use of distributed hash tables for discovering (Web) services is not a new concept. Other related work exists that proposes the discovery of such services through decentralised registries [231]. These distributed hash table implementations often rely on well-known *overlaying* communication layers such as Kademlia, Chord, or Pastry [232]. The protocol layer can be adapted to different implementations. While some protocols only enable communication between peers, protocols such as IPFS [233] or OpenDHT [234] offer public endpoints that enable outsiders to query information.

However, the semantic and processing interoperability of such a gateway limits the accessibility by third parties. These open gateways provide no insights into the hash function that is being used and require prior knowledge of the hashed key itself. Furthermore, these public gateways often provide no semantics on the available functions, which are available and require prior knowledge on how to interface with these gateways.

Linked data inherently already offers the decentralisation of data. With our POSO ontology, we offered interoperable positioning systems with the opportunity to describe themselves on the Semantic Web. Positioning systems are identified by the URI linking to the description of such a system. However, discovering or querying linked data becomes increasingly complex when more sources provide *fragments* of this data. Without additional knowledge, this system cannot be discovered. To address this issue, we propose a solution that offers a distributed

discovery through linked data and facilitates client-side querying through linked data fragments (LDF).

Retrieving or discovering data in our implementation requires no knowledge about our proposed specification and uses the inherited properties of linked data to traverse peers on the Semantic Web. Adding new nodes or data to the network relies on existing specifications or vocabularies such as the TREE hypermedia specification [216], the Solid project [139] and the Schema.org vocabulary [204].

Earlier in Section 4.4, we have already decided on the use of the Solid project for enabling user-centric storage of data produced by positioning systems. Essentially, this entails that each user's Pod could potentially serve as a node in our DHT network.

5.3.2 Architecture

In this section, we provide a high-level architectural overview of our Linked Data Hash Table (LDHT) solution, which is based on the Kademlia DHT protocol. In Sections 5.3.3, 5.3.4 and 5.3.5 we further detail the protocol and its use for discovering positioning systems.

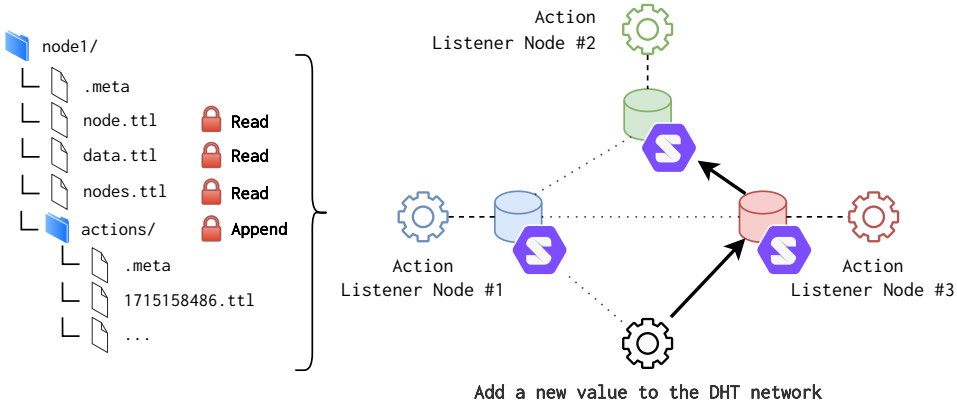


Figure 5.12: Linked Data Hash Table (LDHT) architecture

Figure 5.12 depicts the basic architecture of our LDHT approach. Each node is a Solid Pod with a process or application running in the background. The Solid Pod is structured in a way that provides *data* (i.e., the collection we want to distribute across the network), the other *nodes* that are known by the node (i.e., neighbouring nodes), information about the node itself, and finally an inbox container with *actions* that is used to trigger an action on the node. Other nodes can create new actions on other nodes to add or remove data from the collection. Each node is responsible for a set of data. If an action is received to store data that should not be placed on the node, the action is propagated to the neighbouring nodes.

Data is indexed by a unique key, and each node is responsible for a range of keys. A node contains information about neighbouring nodes and the range of data they are responsible for. When the node wants to look for data indexed by a certain key, it can use this knowledge to traverse the neighbouring nodes to retrieve the data.

Our solution's overlaying network architecture is based on the Kademlia protocol. Kademlia is a decentralised DHT protocol designed for efficient lookup in peer-to-peer (P2P) networks. It structures the network as a binary tree, where each node has a unique identifier (node ID). Each node maintains a routing table to locate other (neighbouring) nodes. The protocol uses XOR distance as a metric to determine the proximity between node IDs, which enables $O(log(n))$ lookups. This XOR-distance is computed as their bitwise XOR and is further detailed in Equation 5.4.

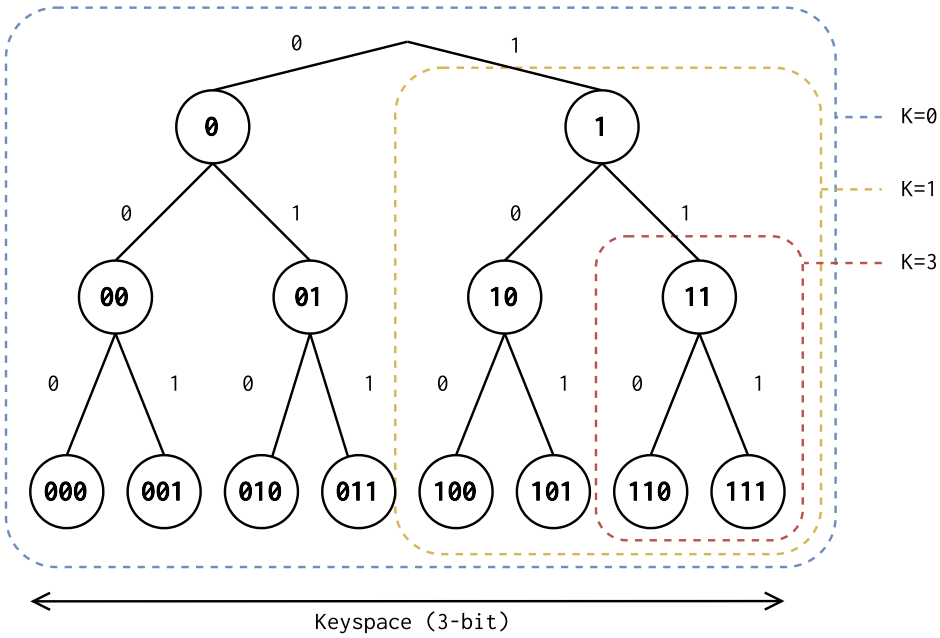


Figure 5.13: Kademlia DHT with a 3-bit keyspace

Figure 5.13 shows a simplified Kademlia DHT with a 3-bit keyspace and 3 buckets. In this example, a total of 8 different keys can be stored. Each node assigns itself a random key.

To maintain routing efficiency, each node stores a routing table structured as k-buckets. A k-bucket holds up to k nodes per distance range. Lookups are recursive and are handled by protocol messages that propagate through the network (i.e., follow the binary tree). The Kademlia DHT protocol consists of four main protocol messages that are sent as Remote Procedure Calls (RPCs):

- **PING**: A ping message is used to verify if a node is still alive. When a node is offline, it should no longer be used in the routing.
- **STORE**: The store protocol message is used to store a key-value pair in a node. This message is sent to the closest node and may be propagated to other nodes in the network.
- **FIND_NODE**: This protocol message is used to request nearby nodes for a particular key. The message is sent to a neighbouring node to request its closest neighbours, and it can propagate through the network.
- **FIND_VALUE**: This protocol message works similar to the searching of a value, but instead of finding a node responsible for a key, it will return the value stored for this key. Just as the protocol message for finding a node, the message can propagate through the network.

Other than the default Kademlia protocol, our architecture only uses an RPC message for storing data or pinging a node. The search for nodes and data is accessible to each node via the Web. Therefore, the node can traverse the *linked data* client-side instead of letting a call propagate through the network.

5.3.3 Hashing Function

To efficiently distribute the data across nodes based on their geographical location, we use a hashing function that maps latitude and longitude values to a unique key in the Kademlia DHT. Latitude has a range from $-90°$ to $90°$ while longitude has a range from $-180°$ to $180°$. The hash function takes the latitude and longitude values as input and outputs a single unique key that falls within the range of keys.

$$\begin{aligned} lat_{normalised} &= latitude + 90 \\ lng_{normalised} &= longitude + 180 \end{aligned} \tag{5.1}$$

As shown in Equation 5.1, we first normalise the latitude (lat) and longitude (lng) to an absolute value. This will ensure that our hashing function produces positive keys.

$$\begin{aligned} partitions &= \frac{size}{\frac{circumference}{360}} = \frac{10}{\frac{40007.86}{360}} \\ latPartition &= \left\lfloor \frac{lat_{normalised}}{partitions} \right\rfloor \\ lngPartition &= \left\lfloor \frac{lng_{normalised}}{partitions} \right\rfloor \end{aligned} \tag{5.2}$$

Next, in Equation 5.2, we *partition* the normalised latitude and longitude into cells of $10 \times 10km$. This essentially splits the Earth in a grid of 8006001 cells that can be distributed amongst all peers. We have chosen this cell size based on the number of expected positioning systems that could exist in a single cell. In our equation, the $size$ is the grid size in km, while the $circumference$ is the Earth's circumference measured through the poles (40 007.87km). This means we require at least a 23-bit node identifier and ideally use 23 k-buckets in each node.

The lookup in a cell would require an $O(n)$ performance, meaning that if our cell size is too large, and therefore covers more positioning systems, it would also require more computation time to look up one particular positioning system. If the cell size is too small, a lot of nodes would exist that have no data included, as the use of indoor positioning systems is currently still limited. The $10km$ grid size was chosen after conducting a set of queries on OpenStreetMap (OSM) to determine the average number of positioning systems for different grid sizes (i.e., indexed by the hashed key). We calculated the average by performing a query that counts the public buildings in a bounding box. This query is executed for each cell in our grid, after which we use the average building count as an average amount of positioning systems that would be available for one hash key. To limit the number of queries to Overpass[8], the API for accessing OSM data, we limited our querying to Belgium as opposed to querying all 8006001 cells. However, this does not account for empty grid cells such as overseas or rural areas.

```
[out:json];
(
  way["building"~"public|government|school|university|
          college|hospital|clinic|fire_station|
          police|library|civic"]
  (49.490271,1.799646,50.390094,2.699469);
);
out count;
```

Listing 5.3: Example Overpass query for a bounding box to count public buildings on OpenStreetMap (with a $100km$ grid size)

Listing 5.3 shows a generated query[9] for retrieving the amount of public buildings in a $100km$ grid cell. In this case, we calculated the average public buildings in a $100km$ cell to be 1605. Similar tests were conducted with different grid sizes until we settled on a default grid size of $10km$ which has an average amount of 16 public buildings and a maximum amount of 403 public buildings in one cell. While we

[8] https://wiki.openstreetmap.org/wiki/Overpass_API
[9] https://github.com/OpenHPS/openhps-dht/blob/29fa15d851037eb31f10e8b730147d0c8d2a919b/test/utils/validator.ts

use this as the default in our `@openhps/dht` module, developers can still override this number for other data collections.

$$hash = latPartition * 31 + lngPartition \tag{5.3}$$

Finally, the hash of the latitude and longitude cell is calculated in Equation 5.3 using the sum of the latitude partition multiplied by the prime number 31 and the longitude. This multiplicated was added to ensure that the hash is unique for each latitude and longitude combination (i.e., when the sum of the latitude and longitude is equal to a previous hash value).

When we want to search for positioning systems at a given latitude and longitude, but the location accuracy for the search is less accurate than $10km$, the lookup will occur over multiple cells that cover the complete latitude and longitude in a radius based on the accuracy.

5.3.4 Storage and Lookup

In our DHT network, we consider a node to be an individual Solid Pod that can contain data. New nodes can join the network and may also leave the network. The DHT network is initialised by letting each node that joins the network generate its identifier using a random latitude and longitude that is hashed using the hashing function. Nodes are located using URIs that link to nodes of the TREE hypermedia specification [216].

When a new node is added to the network via one or more nodes, this change is propagated through the nearby nodes. Each node has 20 buckets that include information about neighbouring nodes based on their proximity. Distance between nodes uses the XOR distance between two nodes as shown in Equation 5.4.

$$distance = \mid nodeID^{otherNodeID} \mid \tag{5.4}$$

$$bucket = \lfloor log_2(distance) \rfloor \tag{5.5}$$

Using the hashing function, a latitude and longitude can be converted to a hash *key*. Each node stores 20 nodes per bucket.

While we can make the data of each node accessible by other nodes, distributed hash tables still require remote procedure calls for storing data and managing nodes. To enable this, we make use of Linked Data Notifications (LDN) [218] and write access to a Linked Data Platform (LDP) container.

```
# Collection of (indoor) positioning systems
<http://purl.org/poso/collection/>  a ldht:Collection ;
  tree:view <> ; # Node described in <>
  # The nodes managing this collection are available in ...
  ldht:nodes <http://purl.org/poso/collection/nodes/> .

# Collection of nodes
<http://purl.org/poso/collection/nodes/> a tree:Collection ;
  tree:view <./nodes.ttl> . # Node described in nodes.ttl

# Description of this LDHT node (ID 37958)
<>  a ldht:Node ; ldht:nodeID 37958 ;
  # Relation to other keys (i.e., node identifiers)
  tree:relation
    [ a tree:GreaterThanOrEqualToRelation ; tree:value 43219 ;
      tree:node <http://othernode1> ; tree:path ldht:nodeID ],
    [ a tree:GreaterThanOrEqualToRelation ; tree:value 37958 ;
      tree:node <./data.ttl> ; tree:path ldht:nodeID ],
    [ a tree:GreaterThanOrEqualToRelation ; tree:value 16161 ;
      tree:node <http://othernode3> ; tree:path ldht:nodeID ];
  # Actions that can be performed on the node
  schema:potentialAction
    [ a ldht:PingAction ; schema:target <./actions/> ],
    [ a ldht:AddNodeAction ; schema:target <./actions/> ],
    [ a ldht:RemoveNodeAction ; schema:target <./actions/> ],
    [ a ldht:StoreValueAction ; schema:target <./actions/> ] .
```

Listing 5.4: DHT node with identifier and relations to the data

Listing 5.4 shows the description of an LDHT node. Each node has an identifier, a list of potential actions that can be performed on the node and relations to the data in the node, as well as on other known nodes. Each node knows (some) of the other nodes in the network, which is available in a separate *collection*. While this separate collection is not needed due to the existence of the `tree:relation` to other neighbouring nodes, it does facilitate defining a collection of all nodes within the network.

In a Distributed Hash Table (DHT) network, finding data involves a distributed search process among the nodes. The process of finding data in our LDHT network does not require the creation of a new action. Instead, the nodes are traversed based on the most likely neighbouring nodes that would contain the data. The traversing and searching for data is done similarly to Kademlia, with the only exception being that the LDHT network can traverse the network itself as opposed to sending a *message* to the nearest node to find the data.

For our hashing algorithm that divides the Earth into cells, the key is first calculated based on the location and search range. For each key, the nearest neighbour is calculated by sorting the neighbouring nodes based on their XOR distance $nodeID^{key}$. The `data.ttl` of the closest node will be checked for the data. If the node is not responsible for this key, its neighbouring nodes are also checked and traversed.

5.3.5 Network Actions

In our linked data approach, network actions such as the addition of nodes make use of Linked Data Notifications (LDN) [218]. These notifications enable a WebSocket connection to a container to automatically receive events whenever a new resource is added to this container.

Information retrieval does not require action as the data is expected to be made public. Any *agent* capable of executing actions will also be able to read the data that the node contains. Due to the nature of our Linked Data approach, even if the backend logic that responds to actions is down, the data contained within the node, as well as pointers to other nodes, remains available as long as the web server is running.

All actions are written as individual resources within a Solid container. Actions are subclasses of `schema.org` [204] and may require a response to indicate that the action has been executed. The method of using *actions* is similar to the use of Solid as a communication broker as detailed in Section 4.4.4. The current draft of the vocabulary is available in Appendix D.

Our decision to use `schema.org` actions as opposed to activity streams [235] was due to the simplicity of the actions and the available vocabulary to indicate the status and type of action. In our set-up using Solid, the activity streams are already used to indicate that a new *resource* has been appended to the container.

Action: `ldht:AddNodeAction`

When a node enters the DHT network, it triggers the addition of a new node action. Listing 5.5 illustrates an example of this action where a new node is added to the network by referencing it using `schema:object`. Actions have a status that indicates if the node has read and completed the action. To avoid race conditions where nodes are waiting for an acknowledgement of their addition, nodes that receive the `ldht:AddNodeAction` will immediately set the status to complete, while the first node waits for confirmation from its neighbouring nodes.

```
# Action to add a new node
<>  a ldht:AddNodeAction ;
  # Date and time of the action
  dct:created "2024-05-18T13:00"^^xsd:dateTime;
  # Current status (active, completed, failed)
  schema:actionStatus schema:ActiveActionStatus ;
  # Publisher of the action (i.e, other node)
  schema:agent <http://node2/node.ttl> ;
  # Node to be added
  schema:object <http://newnode/node.ttl> .
```

Listing 5.5: Turtle markup of an add node action

Action: `ldht:RemoveNodeAction`

Offline nodes will be removed from the network. The remove node action will update the network accordingly. This action is sent by a node that no longer wishes to be part of the DHT network (e.g., due to becoming offline). Other than the addition of a node, the action is immediately resolved once it is received by the neighbouring nodes; there is no need to wait for a confirmation.

Action: `ldht:StoreValueAction`

An action is used to trigger the node to store a new value. Depending on the other nodes that the target node is aware of, the data will be forwarded to other nodes in the network. Neighbouring nodes may not be responsible for storing the new value, but may have more knowledge of other nodes that are responsible. By forwarding the action from one node to the other, the data ends up in the correct node.

```
<>  a ldht:StoreValueAction ;
  dct:created "2024-05-18T13:00"^^xsd:dateTime;
  schema:agent <http://node2/node.ttl>;
  schema:object [ a ldht:Entry ;
    dct:identifier 48665 ; schema:value <http://some_ips/>
  ] ; schema:actionStatus schema:CompletedActionStatus .
```

Listing 5.6: Turtle markup of an action instructing a node to add a new value

In Listing 5.6, an example action is provided for instructing a node to add a new value based on a key (i.e., `dct:identifier`). Depending on the access rights of the collection, the store action can be created by any agent, authenticated or not. It is the responsibility of the node to validate the data.

Action: `ldht:PingAction`

Network actions require backend logic that listens for and responds to these actions. In order to ensure that the backend logic is still operational, a ping action is used to test if the node is active.

```
<>  a ldht:PingAction ;
  dct:created "2024-05-18T13:00"^^xsd:dateTime ;
  schema:agent <http://node2/node.ttl> ;
  schema:actionStatus schema:CompletedActionStatus .
```

Listing 5.7: Turtle markup of a ping action

Listing 5.7 illustrates a ping action created on a node by another node in the network. Initially, the action status is set to `schema:ActiveActionStatus` to indicate that the ping is active and pending. The backend logic, which has write permissions to this action will update the status to completed to indicate it responded to the ping request. If the ping times out, the node that initiated the ping will update the status of the action to `schema:FailedActionStatus` to indicate to the node that it should no longer respond to the ping request if it comes back online.

5.3.6 Implementation in a Positioning System

We envision that an (indoor) positioning system will register itself with a known node. This node can be a self-hosted node by the system itself or a node provided by a third-party service that provides the *poso collection*.

To demonstrate how we envision the use of LDHT for registering a positioning system, we provide an example using OpenHPS in Listing 5.8. The model adds a solid service to the positioning system on lines 11 to 14, which is the same service that is used to store user data in a Solid Pod. The service is responsible for handling authentication and the retrieval and storage of datasets. Next, a `DHTService` is added with a `DHTRDFNetwork`. While our `@openhps/dht` module is primarily aimed at an LDHT implementation, other types of networks can be added in the future.

The LDHT network is initialised with the collection URI on lines 2 to 4. Optionally, a local node can be added to the network. If this is the case, new nodes and data will be sent as an action to this node. When the local node is not set, a new node is created using the authenticated Solid Pod. The `DHTService` is created on lines 10 to 12 using the previously initialised network. To automatically register the positioning system on the network, a geospatial boundary must be set in addition to the URI where the positioning system is detailed.

```
// Create LDHT network
const network = new DHTRDFNetwork(
  'http://purl.org/poso/collection/'
);
network.setLocalNode(
  LocalRDFNode.fromURI('https://.../nodes/poso/node.ttl')
);

// Create DHT service with the LDHT network
const service = new DHTService(network);
service.setBoundary(/* ... */);
service.setSystem("https://my-positioning-system.com/");

// Create a positioning model
ModelBuilder.create()
  .addService(new SolidClientService({
    clientId: "...", clientSecret: "...",
    clientName: 'OpenHPS', autoLogin: true,
  }))
  .addService(service)
  .from().via(/* ... */).to().build();
```

Listing 5.8: OpenHPS implementation of linked data hash tables for adding a positioning system

5.3.7 Conclusion

In Section 5.2, we proposed a solution for the local discovery of positioning systems using beacons that advertise signals to nearby receivers. One of the limitations we discussed in that solution is the requirement for additional infrastructure to merely enable the discovery process. To address this limitation, we introduced the concept of a Linked Data Hash Table (LDHT) network that utilises the Semantic Web and Solid to create a distributed and self-organising network of nodes. The LDHT network allows for the seamless addition and removal of nodes, as well as the storage and retrieval of values within the network.

LDHT nodes are not self-managing, meaning each node requires logic behind the scenes to orchestrate the distribution of data and the linking of neighbouring nodes. We envision that these nodes run alongside a personal data vault (PDV) such as Solid, or are integrated within the PDV.

In our current implementation of this specification, we have implemented the main functionalities for achieving distributed lookup and storage. Future work should expand our implementation to include additional features to ensure replication of data among nodes in case a Solid Pod (i.e., peer) is no longer accessible. While our

solution is generic for a wide range of use cases, not limited to spatial or hierarchical ordered data, we focus on the use case of global discovery of positioning systems. Future work should investigate different scenarios, as well as the scalability of the network on a larger scale.

At the beginning of this chapter, we scoped our research to indoor positioning systems bound to a geographical location. While this is true for most IPSs, environments such as cruise ships can benefit from an IPS, but are not spatially bound to one location. Our LDHT solution would not be feasible in such a scenario. We envision that for these types of use cases, our previously presented SemBeacon solution can offer a more dynamic proximity discovery.

For the human makers of things, the incompletenesses and inconsistencies of our ideas become clear only during implementation.

– Frederick P. Brooks Jr.

Chapter 6

Applications and Technical Evaluations

Throughout our research, multiple prototypes, applications and software libraries were developed to perform technical evaluations on each research question and solution to these research questions. All of our prototypes were developed on top of the OpenHPS framework. While the framework originally handled the creation of positioning systems, it was later expanded to represent these systems and the data they produce in linked data. Using this new expansion, we developed prototypes for decentralising the data produced by these systems. All applications and technical evaluations influenced the design of our solutions and eventually were used as inspiration for the architectural design in our integrated solutions, which will be detailed in Chapter 7.

6.1 Multi-Sensor Ball Tracking

One of our initial prototypes that was developed as part of the OpenHPS technical report [32] is the tracking of a toy ball through multiple sensor streams. The purpose of this prototype was to validate the capability of OpenHPS to handle different data types and positioning systems to validate research question RQ1 and the underlying research questions RQ1.1 and RQ1.2. In addition, we wanted to validate that our design for a process network could handle more complex streams that contain loops and different throughputs for each sensor.

For this prototype, we developed a positioning system for a Sphero Mini toy using the internal sensors of this toy and an external camera. The Sphero provides raw sensor readings for the linear and angular velocity, raw accelerometer data, orientation and an internally computed position. This internal position is computed

by the Sphero toy itself and makes use of the motor velocity, accelerometer and gyroscope.

We make use of the `@openhps/core`[1], `@openhps/opencv`[2] and the use case-specific `@openhps/sphero`[3] modules to construct a model that fuses these multiple sources into one position. The model consists of four sources; the video input, an internally computed position, the input that is sent to the toy and finally the dead reckoned position that is calculated by OpenHPS using the provided velocity.

Our setup is shown in Figure 6.1. We used yellow floor markers to define an area of $260 \times 200\,\text{cm}$. The camera is positioned with a perspective view of the area and the start position of the toy is at the bottom right corner of the camera source. The complete dataset, including the raw video recordings, was published on Kaggle [39].

Figure 6.1: Multi-sensor ball tracking overview

For the scope of this prototype, the toy performs a simple spiralling trajectory. The input for this device consists of an orientation (heading) and speed. Before the start of our input trajectory, we manually calibrated the origin orientation. This

[1]https://github.com/OpenHPS/openhps-core/
[2]https://github.com/OpenHPS/openhps-opencv/
[3]https://github.com/OpenHPS/openhps-sphero/

provides us knowledge on the start position and orientation used by the internally calculated position, which allowed us to define a frame of reference.

Various methods exist to combine the aforementioned positioning methods. Figure 6.2 shows the simplified graph representation of our demonstrated positioning model. Starting from the four different sources on the left of the figure, we will discuss how each signal is processed and fused. We use two feedback loops from our fused position (originating from the sink node on the right) to provide temporal information to our positioning model.

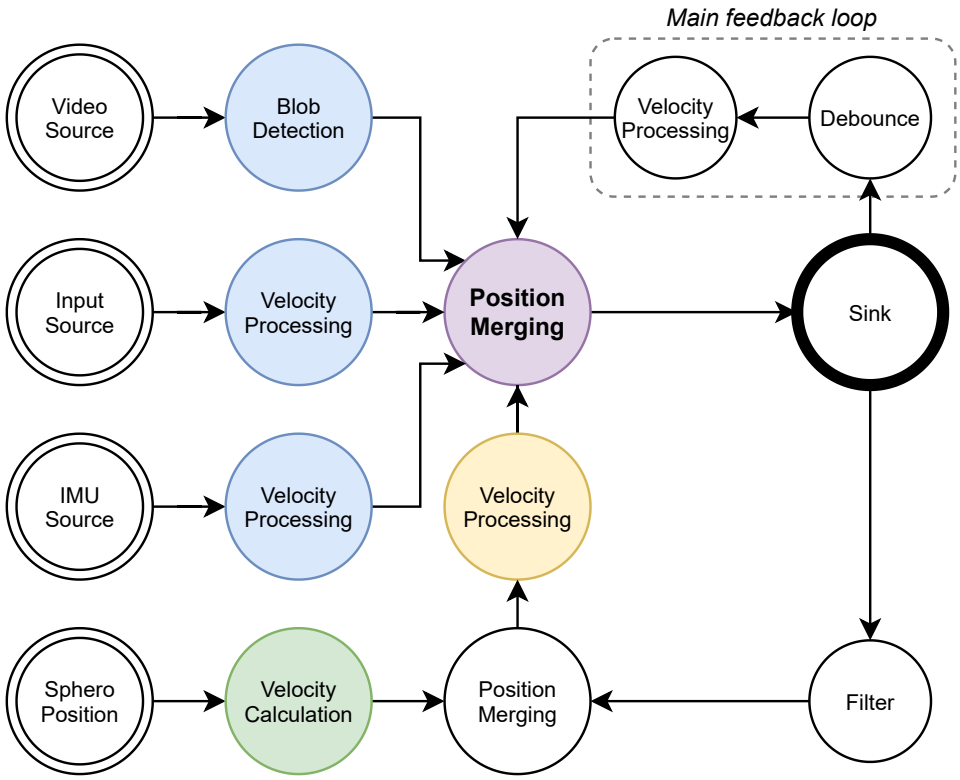


Figure 6.2: Demonstrator positioning model

The positioning result of each independent source is shown in the trajectory plots in Figure 6.3. Each positioning method has a different frequency, resulting in varying amounts of data points used to determine the position. Our main feedback loop indicated on the top right of Figure 6.2 ensures that the fused position never relies on a single source.

6.1.1 Input Control

Input to the Sphero is given using a heading (degrees), speed (0–255) and roll duration. We assume that the Sphero has a maximum speed of 1 m/s as documented on

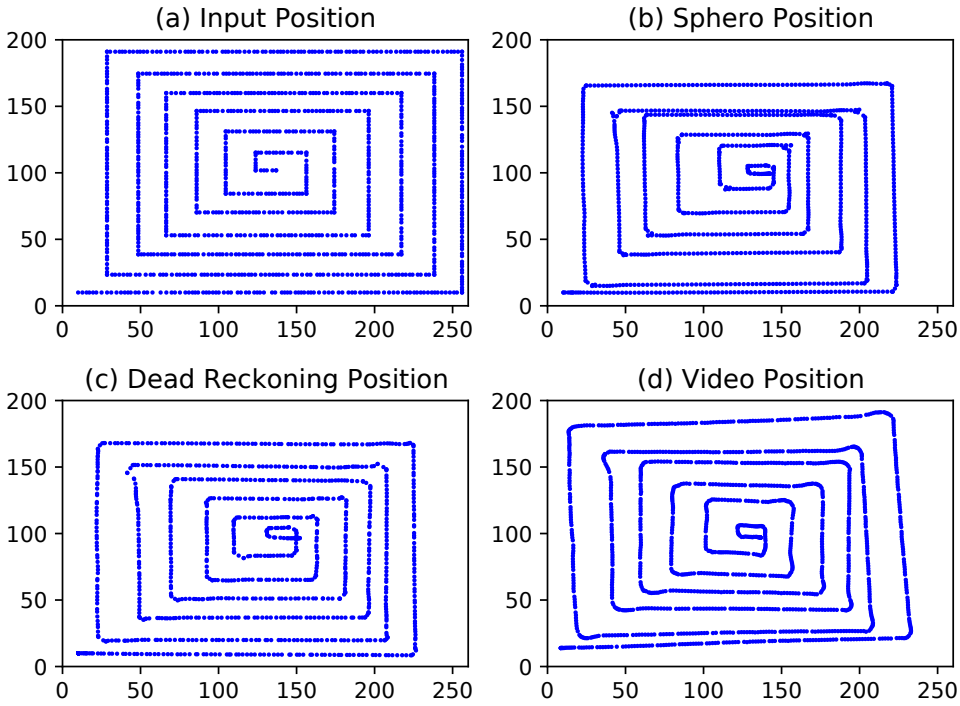


Figure 6.3: Individual position estimates for the given input (a), including the internally calculated position (b), dead reckoning position (c) and video source (d). Units of both the X- and Y-axis are in centimetres

the product website[4]. As input trajectory, we provide a spiralling rectangle starting from an outer corner to the centre of the area with a speed of 150 (= 0.58 m/s). We provide a basic roll duration of 4.2 seconds (= 243.6 cm) for the x-axis and a roll duration of 3.2 seconds (= 185.6 cm) for the y-axis. Essentially, this means that the ball will roll for a specified time, with a speed of 0.58 m/s in a particular direction. Every turn, the duration of the movement along the x-axis is reduced by 168 ms while the movement along the y-axis is reduced by 128 ms, creating a spiralling rectangle. This input is fed to a velocity processing node that is built into the core component of our framework, resulting in the output shown in Figure 6.3a.

6.1.2 Visual Positioning

In our prototype, we wanted to evaluate that apart from regular sensor data, we can also tackle the problem of positioning with visual input through our processing network. The video source uses the OpenCV [186] library to capture a 30 FPS camera feed from the camera which has a perspective view of the floor. When processing the video stream, we create the inverse perspective view by manually specifying

[4]https://sphero.com/pages/support?hcUrl=%2Fen-US%2Fsphero-mini-faq-244300

the position of four yellow markers on the floor. This creates a wrapped image rectangle of $1040 \times 800\ px$.

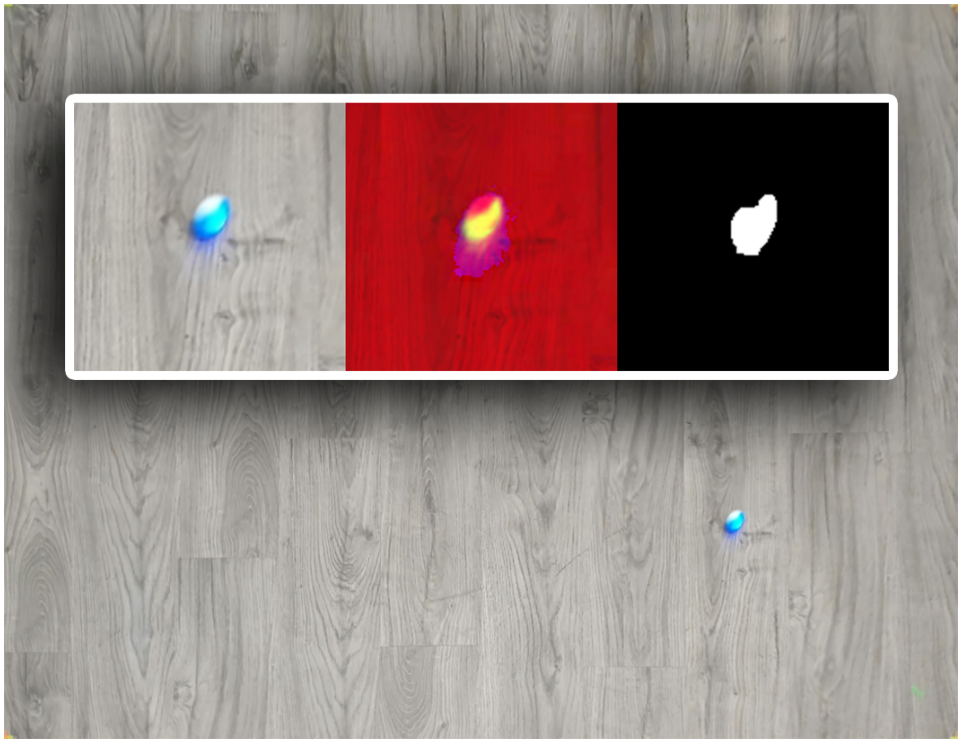


Figure 6.4: Conversion of wrapped video to blob

Once the video source is wrapped, blob detection is used to determine the centroid position of the blue toy. We apply a colour mask that converts the image to an HSV colour space and performs a masking filter to only show the blue ball as illustrated in Figure 6.4. Next, in Listing 6.1 we create a custom processing node that sets the position of our tracked object to the pixel position of the blob (i.e., the centre of the largest contour). As the accuracy for our position, we take the square root of the blob area. A reference space is created (lines 1 to 4 in Listing 6.2) and applied to the output position (pixel coordinate) on line 22 to scale it to the corresponding rectangle dimensions.

Without interference from other sources, the video processing provides the output shown in Figure 6.3d. We will use this source as our most accurate position, as it is the only available external positioning method.

6.1.3 Internal Sensors

The Sphero that we are using for our prototype is aimed towards children who want to learn development in a fun way. To cater to this audience, the Sphero

includes an internally calculated position that is computed and provided to the user for educational purposes. We use this internal position as a source itself to compare and experiment with using this internal position to determine a more accurate position.

In Figure 6.3b we show the internal positioning calculated by the Sphero, converted to a certain reference space created with our calibrated orientation knowledge. Instead of using the raw position, we determine the displacement of this internal position (using the filtered feedback loop shown in Figure 6.2) and apply this displacement to the fused position.

```
class ContourDetectionNode extends ProcessingNode<VideoFrame> {
  public process(frame: VideoFrame): Promise<VideoFrame> {
    return new Promise((resolve) => {
      let contours = frame.image.findContours(
        OpenCV.RETR_EXTERNAL, OpenCV.CHAIN_APPROX_SIMPLE);
      if (contours.length >= 1) {
        contours = contours.sort((a, b) => a.area - b.area);
        const m = contours[0].moments();
        const center = new OpenCV.Vec2(
          m.m10 / m.m00, m.m01 / m.m00);
        const position = new Absolute2DPosition(
          center.x, center.y); // Center as 2D pixel position
        position.unit = LengthUnit.CENTIMETER;
        position.accuracy = Math.sqrt(contours[0].area);
        frame.source.setPosition(position);
      }
      resolve(frame);
    });
  }
}
```

Listing 6.1: Contour detection processing node

Apart from an internally calculated position, the toy provides raw sensor data for the accelerometer, gyroscope, orientation and velocity (internally fused from the motor velocity and acceleration). For the scope of this demonstration, we make use of this velocity and orientation to compute the position using OpenHPS. The output of this source is shown in Figure 6.3c.

6.1.4 Model Creation

With each sensor stream configured, we combine the four streams or graph shapes. These graph shapes include our video output, internal position, input and dead reckoned position. We use a built-in object merging node that merges frames

```
const videoSpace = new ReferenceSpace(defaultSpace)
  .translation(1040, 800)
  .rotation(new Euler(180, 180, 0, 'ZXY', AngleUnit.DEGREE))
  .scale(4, 4); // First transform, then rotate and scale
/* ... */
export default GraphBuilder.create()
  .from(new VideoSource(new CameraObject("sphero_video"), {
    autoPlay: true, fps: 30, throttleRead: true,
    source: new CameraObject("sphero_video")
  }).load("/dev/video2"))
  .via(new ImageTransformNode({
    src: [
      new OpenCV.Point2(307, 120), new OpenCV.Point2(1473, 87),
      new OpenCV.Point2(1899, 891), new OpenCV.Point2(20, 1024),
    ],
    height: 800, width: 1040  // 200 x 260cm
  }))
  .via(new ColorMaskProcessing({
    minRange: [90, 50, 50], maxRange: [140, 255, 255]
  }))
  .via(new ContourDetectionNode())
  .convertFromSpace(videoSpace).to();
```

Listing 6.2: Graph shape video

where the source UID is equal to “sphero”. The merge node will wait until all of its incoming edges push a frame, or the timeout of 20 ms has been reached. By default, this merge will use the weighted average of all incoming positions, velocities and orientations (with the weight being the inverse of its accuracy). Developers that create a positioning system with high-level sensor fusion have the choice to choose their own strategy by, for instance, selecting a single position based on the highest accuracy.

This final fused position is presented in Figure 6.5a. Compared to the individual positioning methods shown in Figure 6.3, we have more data points for our positions. This is because we do not wait for all sources to provide data before computing the next position (20 ms timeout). Our feedback loop called “feedback” ensures that position fusion never relies on just one source and always takes the previously computed position into account.

6.1.5 Evaluation

We have shown our completed positioning system in the previous section. Four sources and a feedback loop resulted in a fused position. In order to evaluate this

positioning model, we removed parts of our video source to simulate an obstacle or blind spots for the camera.

The goal of this evaluation is to first ensure that the positioning model can function with missing information and to determine the error as a result of this missing positioning data. To illustrate a baseline of the remaining sources that will take over the positioning, we show the merged position of all sources except the video source in Figure 6.5b.

Figures 6.5c and 6.5d show two examples with video blind spots (grey areas). Indicated in blue are the data points where the video processing was still able to detect an object, whereas the positions calculated without input from the video source are highlighted in red in the figures.

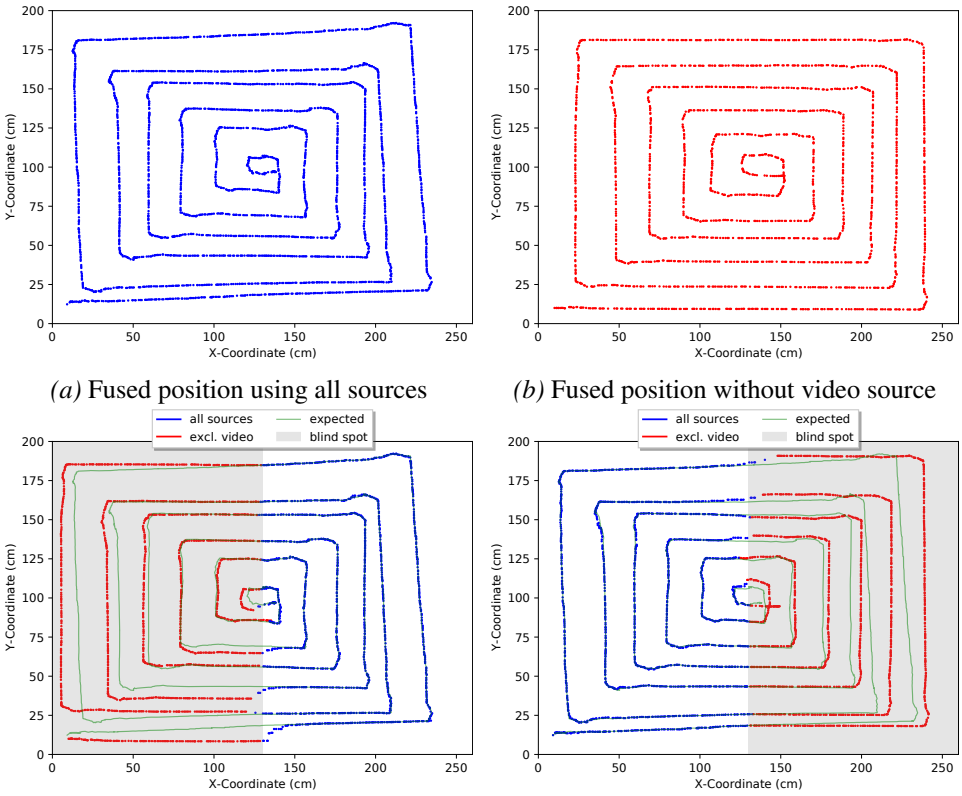
Source(s)	**Avg error**	**Max error**
all sources (Figure 6.5a)	0.00 cm	0.00 cm
input control only (Figure 6.3a)	23.07 cm	50.06 cm
internal position only (Figure 6.3b)	16.16 cm	33.38 cm
dead reckoning only (Figure 6.3c)	17.09 cm	34.44 cm
video source only (Figure 6.3d)	1.30 cm	4.74 cm
all sources excl. video (Figure 6.5b)	13.59 cm	29.73 cm
blind spot left (Figure 6.5c)	4.26 cm	21.65 cm
blind spot right (Figure 6.5d)	4.81 cm	24.40 cm

Table 6.1: Average and maximum XY position error compared to the fused position with all sources

In Table 6.1, we show the average and maximum position error compared to the final fused position from Figure 6.5a. This error is determined by taking 100 timestamped key points in each trajectory (every 51 ms) and calculating the average and maximum difference for those points.

Our results show that the video source is the main positioning method in the fused position. Blind spots in this source result in the model falling back to the remaining dead reckoning. However, the positioning model is self-correcting and will gradually align with the video source position once it becomes available.

The positioning model illustrated in Figure 6.2 is highly adaptable depending on the desired outcome. For example, noise filtering nodes such as a Simple Moving Average (SMA) can be used on video accuracy to provide a smoother transition at the border of the blind spot. With the evaluation in Figure 6.5 and Table 6.1, we have proven that multiple producers of sensor information can be merged into a continuous stream of fused positions. By creating blind spots in our video source, we have shown that the model created with OpenHPS is capable of running without our main visual positioning method.



(a) Fused position using all sources

(b) Fused position without video source

(c) Fused position with camera blind spot on the left

(d) Fused position with camera blind spot on the right

Figure 6.5: Fused positions processed by our model

6.2 Robot Obstacle Detection

Continuing from an earlier research project [236] that started before the work on our OpenHPS framework, we wanted to investigate the possibility of performing basic obstacle detection using OpenHPS in a *Sumo Robot*. A Sumo Robot is a small robot that is placed in a circular area together with other opponents. The goal of the robot is to push the other robot outside the circle while making sure that the robot itself does not drive outside the circle. To help the robot achieve this goal, the border of the arena is equipped with a black line that the robot can detect using sensors to avoid driving off the edge. Our robot is depicted in Figure 6.6.

In a real-world scenario, the algorithms used in these robots in basic competitions are often heuristic and do not require complex computations due to the limited sensor input. Our aim with this experiment was to test the REST API module of OpenHPS, as well as test the possibility of implementing basic, but configurable

Figure 6.6: Custom-designed Sumo Robot with Wi-Fi connection

algorithms as custom processing nodes. This would allow us to easily tweak the algorithms to optimise the robot's performance based on the sensors' performance.

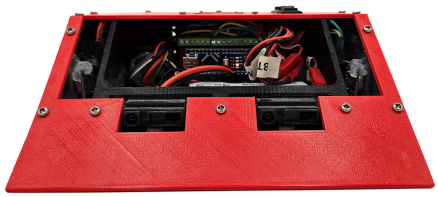
The custom robot we are using has four line sensors as shown in Figure 6.7c, two accurate but narrow front sensors (see Figure 6.7a) and two inaccurate but wide range side sensors as shown in Figure 6.7b. The hardware consists of an Arduino Nano with no network connectivity. It was extended with an ESP8266 module to enable Wi-Fi communication and time synchronisation.

Positioning was done via a server application running a positioning system created with OpenHPS. Using `@openhps/rest`, a web server was created that offered a REST API for transmitting data obtained by the robot via HTTP POST requests. This data contains a snapshot of the current sensor values (encoded as JSON data) and is transmitted two times per second. The slow transmission speed was due to several factors, which we addressed in the framework and are detailed in Section 6.2.2.

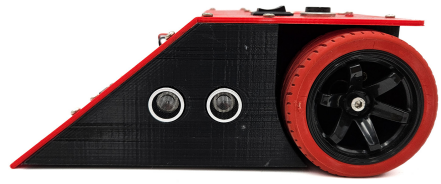
Each sensor value was treated as a source node and was pushed through the positioning model accordingly. Our sink node triggered an HTTP response to the original HTTP request and contained the speed for both motors, essentially acting as the remote controller for moving and rotating the robot.

6.2.1 Positioning Model and Custom Nodes

One of the main concepts in our positioning model was the use of two data objects to represent the main concepts for a Sumo robot; a `LineDataObject` that represents the line and is used to determine if the robot is nearing the edge, and a `OpponentDataObject` that represents the opponent that should be pushed out of



(a) Front of the robot showing two IR distance sensors



(b) Side of the robot showing an acoustic distance sensor

(c) Bottom of the robot showing four line sensors

Figure 6.7: View of all the sensors of the sumo robot

the arena. Depending on whether the line and/or opponent was detected, the robot would perform a certain manoeuvre. Due to the limited available sensor data, data objects (i.e., opponents and lines) were considered stationary and were not tracked in time. However, our positioning model was modified to account for certain types of sensor noise.

Figure 6.8 shows the positioning model with a basic, but modular approach to the processing of data. Line sensor readings were merged into a single node that would determine if a line was detected, and if yes, where in relation to the orientation of the robot it was located. Similarly, opponent sensor readings were merged for the front sensors and side sensors after which both of them were merged in the `OpponentDetectionNode` which concluded if an opponent was detected, and the most probable location of the opponent. Finally, both streams of processed sensor data were merged with the previous historical value and arrived in a sink node that determined the action of the motors (i.e. forward, left or right).

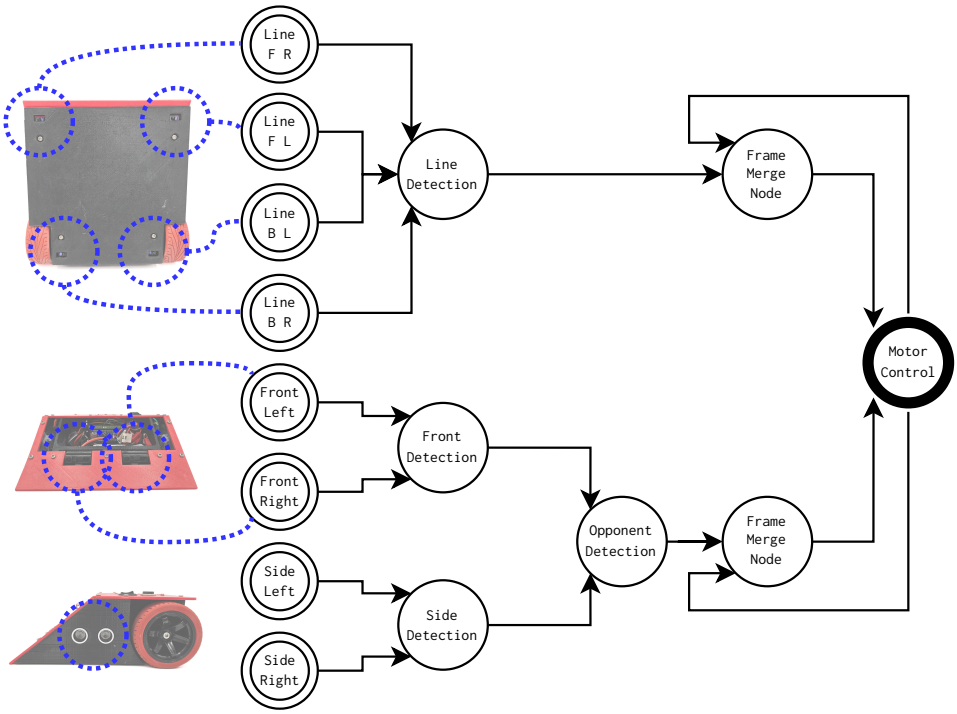


Figure 6.8: Positioning model for sumo robot

Processing Node: `LineDetectionNode`

The first custom node was the line detection node that would create a new data object to represent the *line*. A line was represented as a simple object that had a relative 2D position to the robot and had eight possible positions depending on how many sensors were detecting a line. When no line was detected, the *line* object was not added to the data frame.

The position of a line can either be directly in front, behind, to the left, to the right, front left, front right, back left, or back right of the robot. The position was determined based on which of the four line sensors were detecting the line and their relative positions.

Processing Node: `OpponentDetectionNode`

Opponent detection was done in two stages. In the first stage, the front sensors and side sensors would each attempt to detect an opponent, with the assumption that an opponent could not be both in front and on the sides.

Based on the sensor data of the front and side sensors, an opponent data object was created with a relative 2D position. Using the known specifications of each sensor (i.e. field of view, minimum and maximum distance) we could determine if

an object's 2D position was straight in front of the robot or slightly off to an angle. Finally, both nodes were combined with the front sensor getting the highest priority due to the accuracy of the sensors.

Sink Node: `MotorControlSinkNode`

Based on the input from the line sensors and distance sensors, the first priority of the robot was to remain within the boundaries of the arena by detecting lines and moving away from the line whenever one was detected. Once no lines were detected, the robot would focus on opponents by moving towards the opponents.

Instructions were sent to either go forward or turn left or right to a certain degree. This data would then influence the current position and orientation of the robot. However, while this position and orientation were recorded and used to transform the position of relative objects, it was not used by the positioning model.

6.2.2 Conclusions

Based on this experiment, we discovered several limitations and development issues in the OpenHPS framework that were addressed as a response to this experiment. A lot of design decisions of the API were shaped thanks to this simple but effective experiment.

The limited sensor data of the sumo robot did not allow for the full potential of all features in OpenHPS. Future work can expand the robot to include IMU data which can help in determining the orientation and provide additional dead reckoning data similar to the application in Section 6.1.

1. **Pushing metadata**: One of the issues we experienced was the need to include data alongside data frames that do not *belong* within the data frame itself. In this experiment, the HTTP request that triggered the creation of a data frame expected a certain response that was only given at a sink node. We solved this issue by adding `PushOptions`[5] to each node push. By default, these simply include metadata about the node initiating the push, but it can be extended to include information such as an HTTP request so the sink can respond to this request.

2. **Controlling inlets and outlets**: Initially, the philosophy of OpenHPS was to create an abstraction of regular stream-based frameworks such as Akka by hiding the concepts of inlets and outlets. Instead, OpenHPS was meant to only offer nodes and internally handle the connection of outlets to inlets. However, with custom nodes, it was often desirable that access to inlets (and outlets) was provided.

[5] `https://openhps.org/docs/core/interfaces/pushoptions`

3. **Separation of concerns**: Earlier versions of OpenHPS included various default positioning algorithms in the core component. With this experiment, almost none of these default algorithms were used. This has led us to the refactoring of algorithms such as IMU algorithms to their own module (e.g., `@openhps/imu`[6]).

4. **Relative 2D/3D position**: In older versions of OpenHPS, a relative position was only represented as a distance, angle or velocity to another object. If a coordinate position was given, it was assumed that the position was an absolute position with a *relative reference space*. This essentially required that every object became a reference space when performing relative positioning to this object, which was complicated when the reference frame was a movable object. In later versions, we introduced a `Relative2DPosition` and `Relative3DPosition` which is used to indicate a position relative to another moving object. A reference space is a fixed space that should not move.

5. **Role of data frames**: Initially, a data frame contained raw unprocessed data that would only be moved or used to a data object when processed. In the case of acceleration or velocity, this would imply that the velocity was part of the data frame and influenced the position and orientation of a data object. With non-standard use cases such as this experiment, it would require developers to create a custom data frame to contain the line and distance sensors. In future versions, we modified this behaviour so sensor data was an object on itself that could influence other data objects — preventing the need for custom data frames for each use case.

6. **Overhead of JSON**: With the limited hardware resources available, creating JSON documents for sending sensor data was resource-demanding. We solved this issue by developing protocol buffers that minimise the size of the data.

6.3 Indoor Positioning Server and Application

Some positioning systems require a training phase before the system can be used for regular positioning. This is called the offline phase of a positioning system. Once training is completed, the positioning system enters its online phase, where this training data is used to determine a position. Stream-based processing networks by default cannot simply switch between different states. To solve this particular issue, we introduced the concept of *services* in OpenHPS. As explained in Section 3.8, a

[6] `https://openhps.org/docs/imu/`

(data) service works independently from the processing network to persist data or to compute this data.

Services have various uses, ranging from time synchronisation to handling user actions. With our first prototype for tracking a ball using various sensors, we focused on the evaluation of the effectiveness of representing such a system as a graph-based processing network. The use of (data) services was limited in this initial prototype. Instead, we focused on the use of services to demonstrate how a state can be stored and switched in the stream-based network.

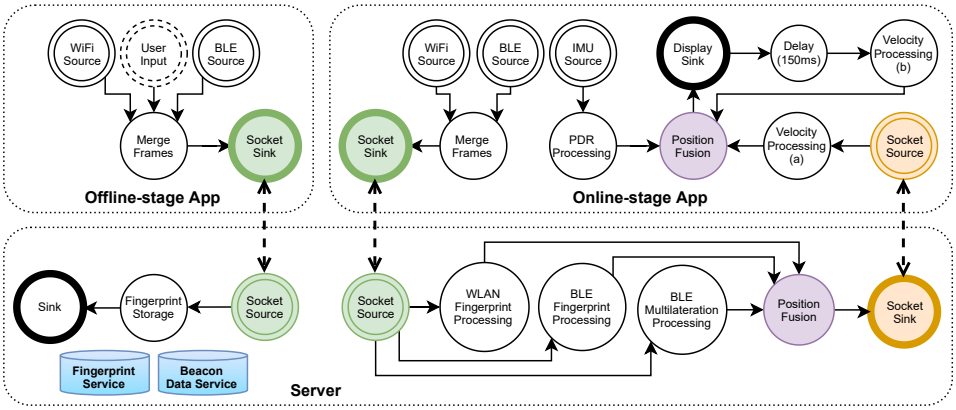


Figure 6.9: Positioning model for server, offline and online application

To demonstrate the use of OpenHPS for offline and online phases of a positioning system, we created a process network with a server, two Android applications and a socket connection for transmitting sensor data to a server and feedback from the server. With this demonstration, we also wanted to showcase the capability of OpenHPS to be used for indoor positioning scenarios as well as the capability of the framework to be used in mobile applications.

The server is implemented based on Node.js[7] and handles the storage of fingerprints and position processing, while the two applications have been developed using React Native[8]. In our `@openhps/react-native`[9] component we provide several source nodes for interfacing with native sensors. Apart from support for React Native, we also added modules for NativeScript, Apache Cordova and Capacitor.

A complete overview of the positioning model is provided in Figure 6.9. Two socket source nodes on the server (indicated in green) handle the server endpoints for the offline and online stage applications. In the offline stage, features from objects within data frames are stored as fingerprints to perform RSSI fingerprinting.

[7] `https://nodejs.org/en/`
[8] `https://reactnative.dev`
[9] `https://openhps.org/docs/react-native/`

The fingerprint service used to store fingerprints is shared with the online stage. For the scope of our evaluation, we used various positioning methods ranging from BLE multilateration and cell identification using 11 BLE beacons, wireless LAN (WLAN) fingerprinting and BLE fingerprinting. A high-level position fusion node fuses the positions based on their accuracy [102]. Finally, the calculated position is sent back to the mobile application through the socket sink node (orange) as indicated in Figure 6.9.

Our mobile application was preconfigured with all locations that required a fingerprint. The user had to click on the location to start the recording of the fingerprint, after which it was presented with instructions on how to hold the phone. With our application, we recorded data in four directions for 20 seconds. After each direction was recorded, the user was instructed to turn to the new location after which the recording started again.

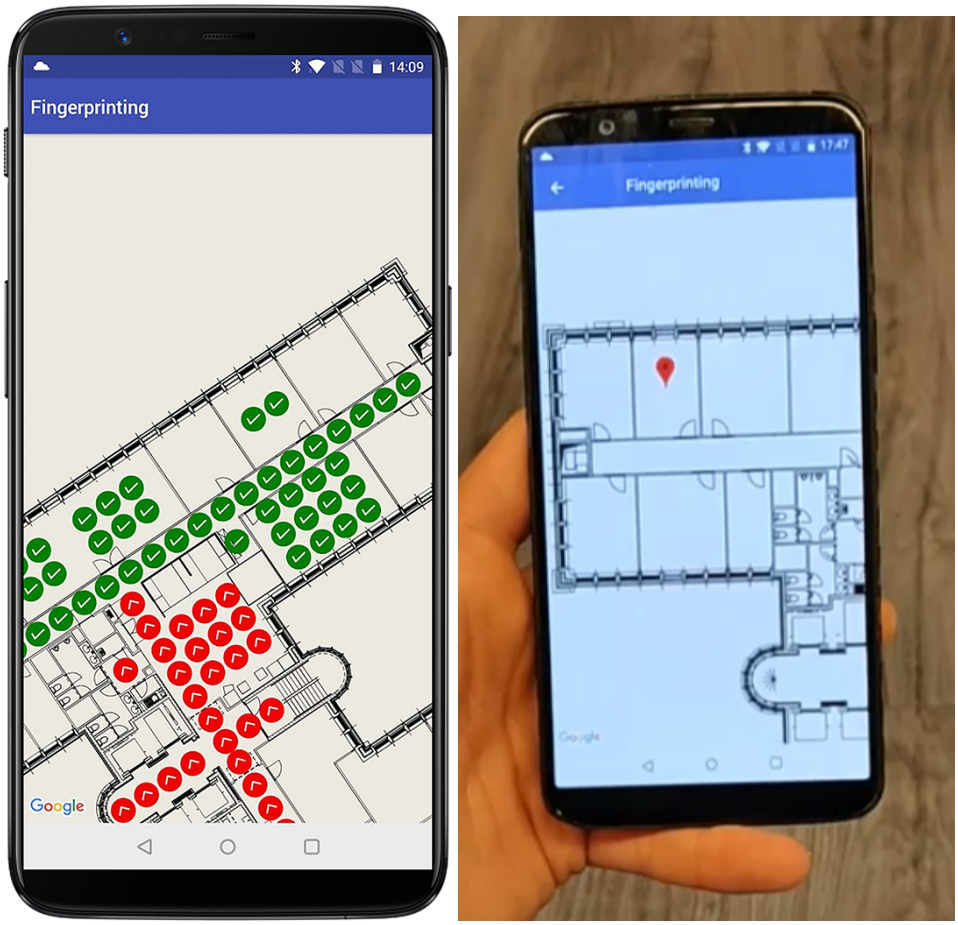


Figure 6.10: Fingerprinting application created with React Native and a screengrab from a video testing the online-phase application

Figure 6.10 shows the mobile application which was used to collect the fingerprints on the left. The application shows the floor plan with all predefined locations that require fingerprint collection. Locations marked green have already been completed while red locations still require the collection of data. After collecting the fingerprints, the data was sent to the server for processing. The photo on the left of Figure 6.10 is a screen grab from a video that demonstrates the online-phase application. In this phase, the user walks around with the mobile device and data is sent to the server for processing. After processing, the position is sent back and displayed to the user as a marker.

6.3.1 Dataset

For the evaluation of our positioning model, we created a fingerprinting dataset of a single floor in the building of our research lab [35]. A visual representation of our dataset is shown in Figure 6.11. The dataset was recorded with a calibration application collecting information from WLAN access points, BLE beacons with a known position (blue) and an IMU sensor.

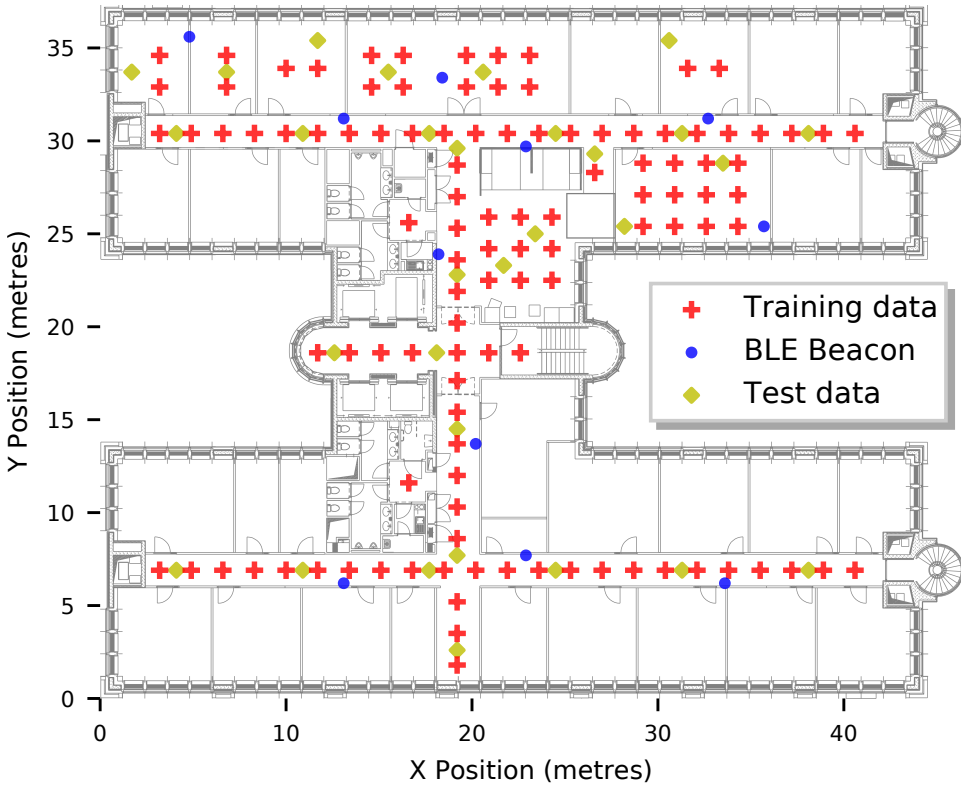


Figure 6.11: Fingerprinting dataset data points

Each of the 110 training data points (red) and 30 test data points (green) were collected in four directions (up, right, bottom and left) by standing still for 20 seconds with a phone held at chest height. In our dataset, we provide the raw data of all WLAN scans, BLE advertisements as well as IMU data including device orientation, acceleration, rotation rate and magnetometer data. Each detected access point is anonymised, but we provide information about SSID groups and the frequency of the access point. This high-dimensional information allows developers to use the dataset to experiment with different fingerprinting techniques that can potentially take the orientation and signal propagation into account for different Wi-Fi broadcasting frequencies [237]. Our methodology for recording this dataset was based on existing datasets and the information that was lacking from those datasets, such as recordings in multiple orientations. We conducted an initial recorded dataset on a smaller scale in 2020 [38]. More information about this initial recording is available in AppendixA.4.

To enrich the contextual information in our dataset, we use the symbolic space abstraction included in OpenHPS to create symbolic spaces for the rooms, corridors two lobbies and toilets. These symbolic spaces will be used to determine the hit rate and are exported as GeoJSON polygonal features. While we provide GeoJSON features of the environment, they were not used to train or influence the behaviour of the tracking algorithms. However, by including these in our dataset, we provide other researchers with the opportunity to make use of this contextual data.

6.3.2 Test Data Points

Our first evaluation of stationary test data points uses WLAN and BLE information to determine the position and symbolic location. For this test, we used aggregated RSSI (received signal strength indicator) results.

We configured the WLAN fingerprinting on our server positioning model shown in Listing 6.3. On lines 2–5, we configure the preprocessing fingerprinting service. The flexibility of our system allows developers to choose how fingerprints are stored, normalised and aggregated. If needed, this service can be replaced with a custom pre-processing algorithm. Lines 15–17 show the creation of an offline fingerprinting node. This is an object processing node extracting features of objects and storing them in the fingerprinting service. The online stage (lines 8–12) can use processed fingerprints to obtain a position. In this particular test, we used a weighted k-NN algorithm that is configured similarly to the parameters used by RTLS@UM in the EvAAL competition of 2015 [238]. The use of general parameterised processing nodes offers great flexibility for developers to tweak the positioning system. In this evaluation, we removed the pedestrian dead reckoning and feedback loop from our positioning model.

```
ModelBuilder.create()
  .addService(new FingerprintService(
    new MemoryDataService(Fingerprint), {
    classifier: "wlan", defaultValue: -95
  }))
  .addShape(GraphBuilder.create() // ONLINE MODE
    .from(/* ... */)
    .via(new KNNFingerprintingNode({
      weighted: true,  k: 4, classifier: "wlan",
      weightFunction: WeightFunction.SQUARE,
      similarityFunction: DistanceFunction.EUCLIDEAN
    })).to(/* ... */))
  .addShape(GraphBuilder.create() // OFFLINE MODE
    .from(/* ... */)
    .via(new FingerprintingNode({
      classifier: "wlan"
    })).to(/* ... */))
  .build();
```

Listing 6.3: Positioning with fingerprinting parameters

Table 6.2 shows the average, minimum and maximum error for different positioning techniques, along with the standard deviation and failed points. Failed points indicate test reference points for which the positioning technique was not capable of determining a position. In the case of BLE positioning, these are the calculated positions where not enough BLE beacons were in range. The symbolic space hit rate represents the amount of test data points that were assigned to the correct symbolic space (i.e., room or zone).

The modularity of our OpenHPS framework allows developers to rapidly adapt and test the sensor fusion for a specific use case. Developers can decide which metrics they want to optimise. Other than simply trying to determine the most accurate position, the goal of a positioning system might be to get the most accurate hit rate, increase the update frequency or minimise energy consumption. In our chosen sensor fusion, we adapt the accuracy of a certain positioning technique based on the available information (e.g., BLE beacons in range) to achieve a higher symbolic space hit rate. Such a sensor fusion algorithm is useful if a positioning system wants to achieve a reliable general location of persons or objects without requiring precise tracking of the location within a small area. WLAN fingerprinting offers the best average error overall, but by adding BLE beacons in symbolic spaces, we can increase the symbolic space hit rate with a minimal impact on the average error.

Positioning Techniques		
WLAN Fingerprinting (k=4)	**failed points**	0
	average error	1.23 m
	minimum error	0.01 m
	maximum error	4.77 m
	standard deviation	1.04 m
	symbolic space hit rate	95.82%
BLE Fingerprinting (k=3)	**failed points**	6
	average error	3.23 m
	minimum error	0.17 m
	maximum error	15.39 m
	standard deviation	2.69 m
	symbolic space hit rate	80.83%
BLE Multilateration	**failed points**	12
	average error	4.92 m
	minimum error	0.74 m
	maximum error	19.26 m
	standard deviation	3.50 m
	symbolic space hit rate	52.50%
Sensor Fusion (WLAN + BLE)	**failed points**	0
	average error	1.37 m
	minimum error	0.01 m
	maximum error	9.75 m
	standard deviation	1.26 m
	symbolic space hit rate	96.67%

Table 6.2: Average, minimum and maximum x/y position error compared to the fused position

6.3.3 Trajectories

In addition to stationary data points, we included several trajectories in our dataset. These trajectories include IMU data alongside the WLAN and BLE data. The process network in our online application sends the WLAN and BLE data to the server, where it is processed similarly to the test data points in Section 6.3.2, while the IMU data is used locally in the application to perform pedestrian dead reckoning. Trajectory sensor information was collected by keeping the phone at chest height while performing the trajectory at a normal walking pace. Other than the stationary points, the update frequency and accuracy are more important than the symbolic hit rate.

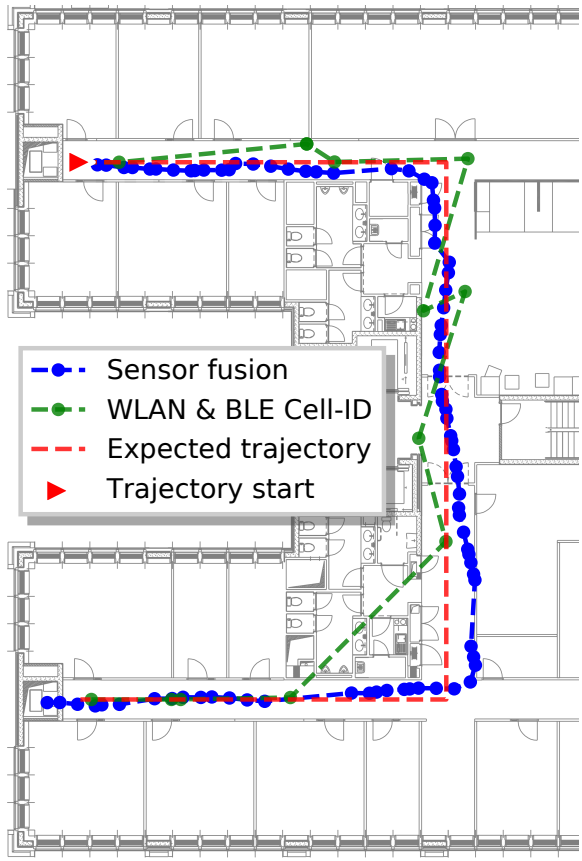


Figure 6.12: Test trajectory with WLAN, BLE and IMU data

The expected trajectory is shown in red in Figure 6.12. We determined the error by comparing the last known position with the actual expected position in the trajectory. While WLAN positioning and BLE cell identification can show a visual representation of the complete route, it only consists of 13 data points that are not synchronised with the user's real-time position. This delay is due to the scan duration and the processing time on the server.

In Table 6.3 we show the maximum and average error for our test trajectory with and without IMU data. The delay caused in the fingerprinting in combination with the slow update frequency causes a larger error compared to the real-time position during the trajectory. Note that the flexibility of OpenHPS allows developers to experiment with different positioning algorithms and fusion techniques to further optimise the system.

The modularity of our framework allows developers to rapidly adapt and test the sensor fusion for a specific use case. Other than trying to determine the most accurate average error, the goal of a positioning system might be to get the most

Positioning Techniques		
WLAN + BLE	**average error**	3.29 m
	maximum error	9.60 m
	standard deviation	2.09 m
	average update frequency	3.04 s
WLAN + BLE + IMU	**average error**	1.26 m
	maximum error	3.10 m
	standard deviation	0.77 m
	average update frequency	0.52 s

Table 6.3: Sensor fusion comparison for test trajectory

accurate hit rate, increase the update frequency or minimise energy consumption. In our chosen sensor fusion, we adapt the accuracy of a certain positioning technique based on the available information (e.g., BLE beacons in range) to achieve a higher symbolic space hit rate. This makes OpenHPS an ideal prototyping solution by simply adding, removing or configuring different nodes.

6.4 Collaborative Positioning Systems

For our paper, *"A Solid-based Architecture for Decentralised Interoperable Location Data"* [43], we developed a set of applications that demonstrate the use of personal data vaults for user-centric data storage of location data. To evaluate our solution with Solid Pods, we used these prototypes of positioning systems which store the information within a Solid Pod[10].

Providing user-centric data storage enables positioning systems to work in a decentralised and collaborative manner. While our storage is centralised around the user, the data from multiple users is decentralised on the Web rather than centralised around the positioning system(s) obtaining this data.

We implemented a non-trivial prototype that uses our user-centric storage with a basic indoor positioning system, one outdoor positioning system and a consumer application that shows a user's recent data along with additional information. With these implementations, we wanted to demonstrate how a user's personal position, orientation and velocity are stored, how applications gain access to this data and how our solution might benefit tracked users. These prototypes helped us to adapt our solution to a more scalable system that can handle a large amount of data. Our final adapted solution was detailed in Section 4.4.

In addition to a validation of our user-centric storage, this proof of concept was also used to verify that the Sensor, Observation, Sample and Actuator (SOSA)

[10] https://github.com/OpenHPS/ipin2022-solid/

and Semantic Sensor Network (SSN) ontologies, detailed in Section 4.2, could be leveraged to describe positioning systems. We used these ontologies before the design of our own POSO ontology with minimal modifications to verify the general descriptiveness of SOSA and SSN.

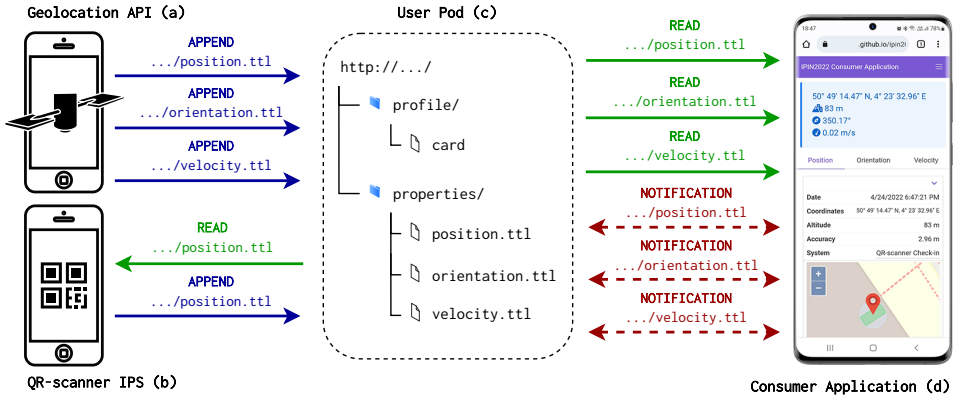


Figure 6.13: Demonstrator with two positioning systems (a, b) and one consumer application (d) connected to the same user pod (c)

Figure 6.13 shows the technical setup of our demonstrator. For the scope of this demonstration, the used indoor positioning system is a QR code *check-in* system that will update the position of a user depending on the check-in and check-out scanning (see bottom left of Figure 6.13). When a user scans the check-in QR code, their location will be updated to the room that they are checking into. When they scan the check-out QR code, their location is updated to the hallway. The outdoor positioning system uses the Geolocation API [64] to update the position, orientation and velocity as depicted in the top left of Figure 6.13. Finally, the consumer application is depicted on the right, which displays the user's positioning data.

Figure 6.14: VUB QR-code check-in and check-out [239]

Our QR-scanning is based on a common use case during the COVID-19 pandemic. As depicted in Figure 6.14, visitors were requested to check in when entering an indoor environment such as our workspace and scanning the check-out QR-code when leaving the area.

```
@prefix profile: <http://.../profile/card#> .
@prefix example: <http://purl.org/ipin2022-solid#> .

:1646891100 a sosa:Observation;
  sosa:hasFeatureOfInterest profile:me;
  sosa:usedProcedure example:qrscanner_checkin ;
  sosa:hasResult [ a geosparql:Geometry;
        geosparql:asWKT """
          <http://www.opengis.net/def/crs/OGC/1.3/CRS84>
          POINT Z(5.893628199 46.641890130 15)
          """^^geosparql:wktLiteral ;
        geosparql:coordinateDimension 3 ;
        geosparql:hasSpatialAccuracy [ a qudt:QuantityValue ;
              qudt:numericValue 598 ;
              qudt:unit qudt-unit:CentiM ] ];
  sosa:observedProperty <> ;
  sosa:resultTime "2022-03-10T06:45:00.000Z"^^xsd:dateTime .
```

Listing 6.4: Example position observation (`position.ttl`)

While in this demonstration we focus on positioning data, the SOSA ontology enables the modelling of properties and observations of any sensor data that belongs to the tracked entity. Properties such as relative positions, raw signal strengths from beacons or raw IMU sensor data can also be modelled similarly as shown in Listing 6.4 with a different value for `sosa:hasResult`. Finally, in Section 6.4.3 we demonstrate how another positioning system or application can use the semantic context of the two positioning systems to predict a more reliable output.

6.4.1 Properties

Each tracked entity should have a set of specific *observable* properties such as a position, orientation, velocity or any other data that a positioning system can provide. These properties indicate information, belonging to an entity, that can be observed by a sensor.

In our demonstration, we manage three properties of the entity that we are tracking: the position, orientation and velocity. Our Geolocation API provides all three properties, while the QR scanning only updates the position property. For the scope of this demonstration, the velocity and orientation are expressed as a one-dimensional value in the `sosa:hasResult` of the example observation in Listing 6.4.

Properties can be discovered by reading or querying the profile card of a Solid Pod[11]. Listing 6.5 shows an example profile card describing a person and *feature of interest* (i.e, `sosa:FeatureOfInterest`), which are accessible URIs to the datasets containing the observations of these properties.

```
:me a schema:Person, sosa:FeatureOfInterest, foaf:Person ;
  vcard:bday "1995-03-10"^^xsd:date ;
  vcard:fn "Maxim Van de Wynckel" ;
  vcard:hasAddress [ ... ] ; vcard:organization-name "VUB" ;
  ssn:hasProperty </properties/orientation.ttl>,
    </properties/position.ttl>, </properties/velocity.ttl> ;
```

Listing 6.5: Example profile card with properties

All position observations from every positioning system are stored in the same resource (`*.ttl` file[12]). These timestamped observations include information on the system that observed them and the procedures used by that system. In the case of the indoor positioning system, it also includes a reference to the room where a check-in occurred.

Later evaluations and work proved that the storage of all observations in a single resource is not beneficial for performance and privacy reasons. In the proposed integrated solution of this dissertation, we will take these issues into account.

6.4.2 Applications

Figure 6.13a shows the Geolocation API application that only appends new observations to the user pod. This type of system only requires the *submitter* role to append new information. The QR code scanner application determines a check-in or check-out depending on the current position and therefore requires read permissions. If the last position is inside a room, scanning the QR code again means the user leaves it. The minimum required role for this system is a *poster*. For the application that visualises the data, we only require the *visitor* role with read permission.

Figure 6.15 shows the authentication and visualisation of the data in the Pod. A user starts by selecting their Pod issuer, after which they are redirected to the provider's login page to request access. After access has been granted, the application can query the last positions, velocities and orientations. Additional information that is referenced within the observations, such as the room, will be fetched from the public triple dataset semantically explaining the shapes and locations of the

[11] `http://ipin2022.solidweb.org/profile/card`
[12] `http://ipin2022.solidweb.org/properties/position.ttl`

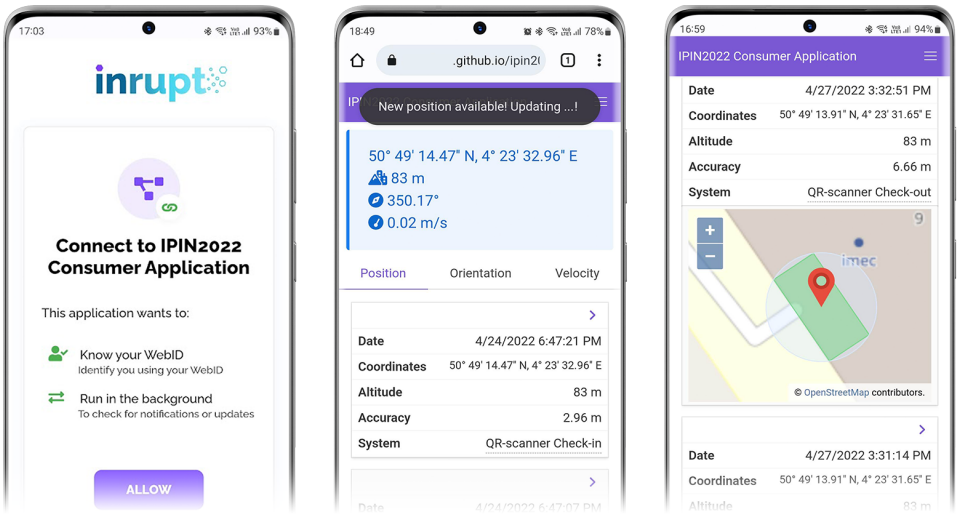


Figure 6.15: Login authentication example and consumer app

rooms[13]. Users of a Solid-based application only need to remember a single login, which benefits the use of an IPS that does not require an additional account.

6.4.3 Querying

In the current Solid specification, the server is not required to handle SPARQL queries due to its computational complexity on the server. Therefore, not every storage provider supports it, preventing us from depending on its existence. In our example implementation[14], we use the Comunica [241] SPARQL engine that supports local queries on fetched datasets as well as server-side querying on supported Solid providers. The engine supports link traversal-based query execution [242] where the set of documents will continuously expand by following the URIs within the dataset that one performs the query on. This allows us to perform queries without previous knowledge of the datasets.

Listing 6.6 shows the SPARQL query used by the consumer application to obtain the last 20 positions, the time when they were observed and the accuracy in metres. With this example, we want to highlight the advantage of additional semantic context and linked data provided by the used vocabularies. Our query is executed on the dataset of the profile where the properties are referenced. The property URIs mentioned in the profile card link to other datasets that contain the observations that we aim to query. These datasets linked within the profile will be automatically

[13]https://purl.org/ipin2022-solid

[14]One solution to this problem is the creation of pre-defined SPARQL queries that offer an endpoint to retrieve that information [240]

fetched by using link traversal. A GeoSPARQL 1.1 query function is used for converting the well-known text literal to a GeoJSON format (line 9) to immediately use the output in our application. We also use additional semantics from the QUDT ontology to convert the accuracy to a base unit regardless of the unit used in the data (lines 12 to 16).

```
1  SELECT ?posGeoJSON ?datetime ?accuracy {
2    ?profile a sosa:FeatureOfInterest ;
3             ssn:hasProperty ?property .
4    ?observation sosa:hasResult ?result ;
5                 sosa:observedProperty ?property ;
6                 sosa:resultTime ?datetime .
7    ?result geosparql:hasSpatialAccuracy ?spatialAccuracy ;
8            geosparql:asWKT ?posWKT .
9    BIND(geof:asGeoJSON(?posWKT) AS ?posGeoJSON)
10   ?spatialAccuracy qudt:numericValue ?value ;
11                    qudt:unit ?unit .
12   OPTIONAL { ?unit qudt:conversionMultiplier ?multiplier }
13   OPTIONAL { ?unit qudt:conversionOffset ?offset }
14   BIND(COALESCE(?multiplier, 1) as ?multiplier) # Default 1
15   BIND(COALESCE(?offset, 0) as ?offset) # Default 0
16   BIND(((?value * ?multiplier) + ?offset) AS ?accuracy)
17 } ORDER BY DESC(?datetime) LIMIT 20
```

Listing 6.6: SPARQL query on the profile card dataset

As shown in Figure 6.13, the consumer application listens for update notifications on the properties. These are used to obtain live updates of the position. In a positioning system, this could be used as a push-based source for sensor and location data. With our example implementation in Figure 6.15, we visually showcase these position changes as a pop-up notification.

6.5 Fingerprint Accuracy Prediction Using CNN

In Chapter 4, we have discussed the importance of semantic and syntactic interoperability to enable multiple positioning applications to work together. However, another important aspect that must be available is the actual data and, more importantly, its accuracy. Developers of positioning systems have the best knowledge about the limitations and strengths of their algorithms. Making this knowledge available for other systems facilitates collaboration.

One of the design goals of our POSO ontology is the ability to describe the processes of a positioning system, so other systems or applications can reason on the accuracy

of the output. This, in turn, helps other services to decide on the importance of the data when performing sensor fusion. Describing such processes and reasoning about their relevance does require a lot of background knowledge of the functioning of such algorithms, which is not always feasible.

In this experiment, we used the OpenHPS TensorFlow (`@openhps/tf`) module to create a neural network that can predict the accuracy of a location, given an RSSI fingerprint. Our design for our neural network is based on CCpos [97], which focuses on using a neural network to determine a position from a set of RSSI values. In our experiment, a regular fingerprinting algorithm such as k-NN fingerprinting [96] is used and compared to real-world positions to determine the accuracy. Our research hypothesis in this experiment is that the neural network approach allows us to determine accuracy based on the available or unavailable fingerprint data. It should also enable us to predict which areas will result in a lower accuracy prediction.

With this experiment, we want to evaluate whether we can use traditional neural network shapes inside processing networks. We also want to investigate if we can reuse parts of this network as individual nodes in a processing network. The challenge with this experiment is that a neural network behaves differently from a processing network. With the process network, we transmit data at runtime from one node to the next by wrapping it in a data frame. In a neural network, this would result in a large overhead of packing and unpacking data. To solve this, OpenHPS will detect neighbouring TensorFlow processing nodes at runtime that are linked together.

Similar to CCpos our neural network is composed of a Collaborative Deep Denoising Autoencoder (CDAE) autoencoder and a Convolutional Neural Network (CNN). The input and output of the autoencoder are the RSSI fingerprints, while the output of the CNN is a single value for the accuracy in metres. The CNN also has an input for the X and Y values of the calculated position. For training and testing, we continued using the dataset introduced in Section 6.3.1 which consists of training and test data points of a single floor in the building of the lab.

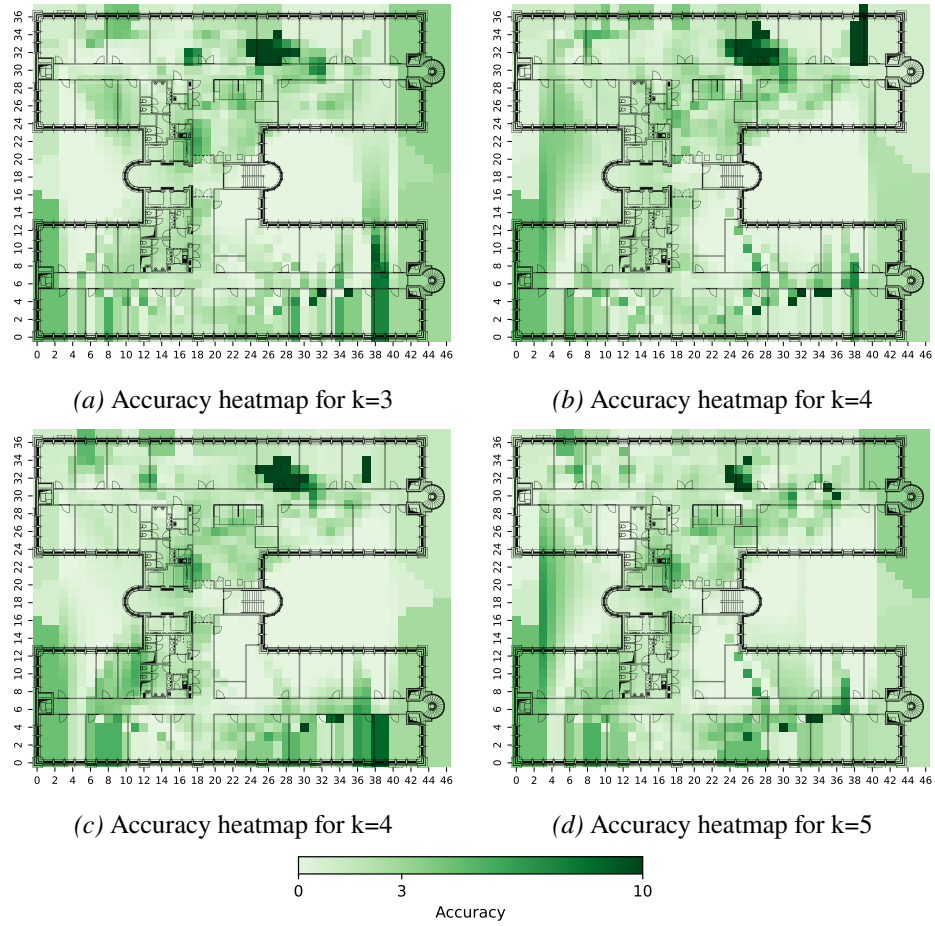
6.5.1 Training

Training of the neural network is done in two parts. In the first part, the training of the CDAE is performed by setting the RSSI values of measured data points as the input and output of the autoencoder. After the training, the decoder is removed and the encoder is used to *compress* the RSSI values to a latent space that aims to capture the important features of the set of RSSI values.

In the second part of the training, the encoder is connected to the CNN network and training is only enabled for the CNN part. The input for the CNN consists of the compressed RSSI values from the encoder along with the X and Y position values from our fingerprinting algorithm.

On a set of data points, we have the known position and the corresponding RSSI fingerprints. This data is used to train the neural network to predict the accuracy of a location based on the RSSI values.

6.5.2 Testing



(a) Accuracy heatmap for k=3

(b) Accuracy heatmap for k=4

(c) Accuracy heatmap for k=4

(d) Accuracy heatmap for k=5

Figure 6.16: Interpolated accuracy heat maps for different k values in the k-NN fingerprinting algorithm. Units of both the x and y-axis are in metres

Figure 6.16 shows a heat map of the predicted accuracy at certain locations according to our model. Data was interpolated and extrapolated from the test data to visualise areas on the floor where fingerprinting accuracy might not be ideal. A higher number of the accuracy (in metres) means the fingerprints in that area produce an inaccurate result.

Based on the predicted accuracies, we could determine how close the predicted accuracy matches the actual accuracy. Such a prediction can help to quickly

determine the performance of different parameters for the fingerprinting algorithm in different areas. Alternatively, it can provide feedback to the user who sets up the system to collect more fingerprints at those locations.

After creating the accuracy heat maps, we noticed inaccuracies in a certain area (top right, next to our lab). We further analysed the cause of this inaccuracy together with our non-anonymised dataset, which we used for our fingerprints. We concluded that the data points collected in front of this office contained more movement (based on the IMU data). Our dataset was recorded on a Sunday to minimise interference from other people moving around the building. However, the office was still occupied on that day, which most likely resulted in both interference as well as a different posture when standing in front of the door to collect the data. Our non-anonymised data also contained various Wi-Fi and Bluetooth devices within the office that were filtered out in the pseudonymised dataset. Insights such as this would not have been easily detected without our accuracy prediction algorithm.

6.5.3 Future Work and Conclusions

With this experiment, we first and foremost investigated the use of OpenHPS for including neural networks within the process models. To do so, we designed a proof-of-concept neural network for predicting the accuracy of the built-in OpenHPS fingerprinting algorithm based on CCpos [97]. We demonstrated that we could predict the accuracy of location estimation with reasonable accuracy. Our model is parameterised, which allowed us to test the predicted accuracy for varying numbers of selected neighbours in our k-nearest neighbours algorithm.

This type of accuracy prediction could be combined with other contextual information in a hybrid positioning system to more efficiently determine which algorithms or parameters of the algorithms to use in different areas. Furthermore, an accuracy prediction can help to provide more context to the output data. However, the process of obtaining this prediction should also be semantically described, which is challenging when working with machine learning, as the process itself is not always easy to describe.

The experiment also helped us expand the OpenHPS framework's inlets and outlets. In our robot obstacle detection experiment in Section 6.2, we indicated that we required additional control for inlets and outlets. The TensorFlow experiment strengthened this requirement since we have to access the inlets to combine layers.

Additionally, this experiment also enabled us to strengthen the monotonicity requirement in OpenHPS. Before, nodes often computed both the position as well as the accuracy of this position. With this experiment, we designed a solution where one node computes the position and another node independently calculates the accuracy of the position.

6.6 Discoverable IoT Devices and Environments

Our solution to local discovery was the design of a Bluetooth specification called SemBeacon. This specification, detailed in Section 5.2, was primarily used to discover positioning systems and their environment.

However, to ensure that our solution was not too domain-specific, we also investigated other use cases where SemBeacon could be leveraged to discover data. One of those use cases was the discovery of Internet of Things (IoT) devices and how to interact with them. Since we are interested in interacting and discovering nearby devices, we focused on the use of Bluetooth Low Energy (BLE) beacons for this purpose.

To demonstrate the deployment and use of SemBeacons, we developed an application[15] that can scan for iBeacons, Eddystone beacons, AltBeacons and SemBeacons. The application will retrieve the environment information that is broadcast by SemBeacons, together with the devices and deployed positioning systems in these environments. The application, along with the SemBeacons was tested in a real-world environment.

Our prototype SemBeacons that were deployed within the building were designed using an ESP32-S3 microcontroller (see Appendix C for more details). The Arduino code for creating and configuring a SemBeacon using an ESP32 can be found on GitHub[16]. Our scanner application is developed in Ionic Capacitor[17], which also allows SemBeacons to be discovered via Web Bluetooth Scanning [183] as well as using an Android or iOS device. A native Android library for scanning beacons is available on the SemBeacon GitHub[18].

6.6.1 Dataset

For our demonstrator, we transformed and extended an existing indoor positioning dataset [35] to semantic RDF data[19]. We redeployed the beacons of the dataset in the same building but replaced two of the original beacons that are closest to the entrance of the floor with SemBeacons (i.e., `BEACON_07` and `BEACON_08`). Other than existing indoor positioning systems for navigating in indoor spaces or tracking the location of physical objects, our solution does not require prior knowledge of the beacons within this dataset. We can publish the semantic data online and let the two SemBeacons broadcast their resource identifier. Any changes to the environment,

[15] https://github.com/SemBeacon/sembeacon-app/
[16] https://github.com/SemBeacon/sembeacon-arduino-esp32/
[17] https://capacitorjs.com
[18] https://github.com/SemBeacon/sembeacon-android-library/
[19] https://sembeacon.org/examples/openhps2021/beacons.ttl

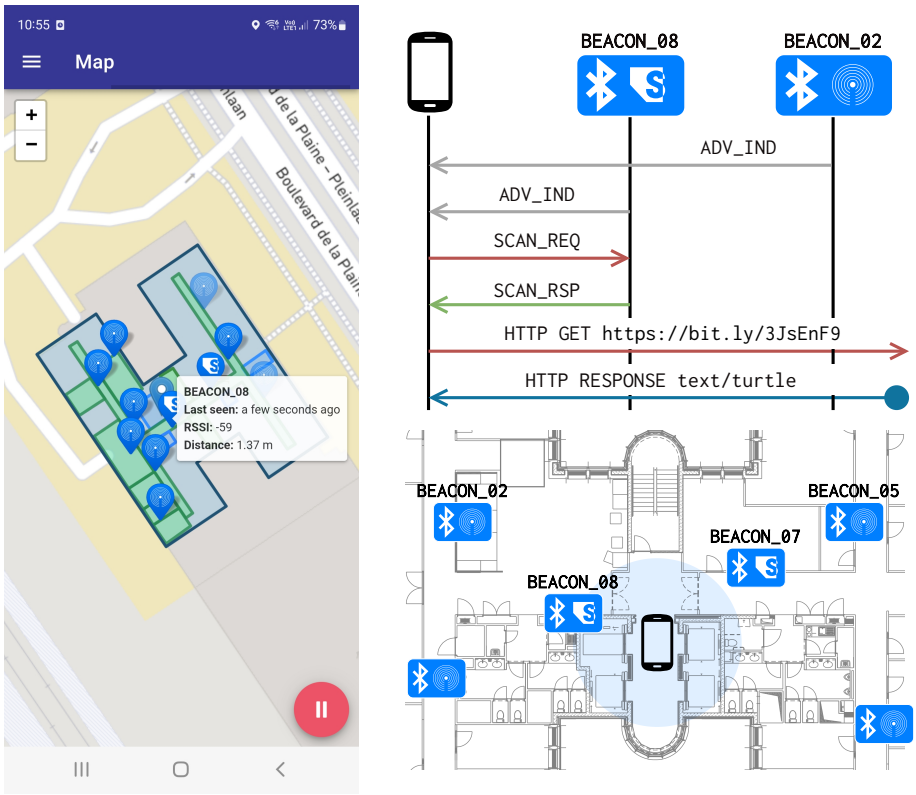


Figure 6.17: Example scenario using the floor plan and transformed dataset of [35]

such as new smart devices or additional details to the geospatial description, can be modified online without the need to update the application or reconfigure the beacons. An example of this transformed dataset is shown in Listing 6.7.

6.6.2 Device and Environment Discovery

We have developed a demonstrator application that is continuously scanning for SemBeacons while showing the user a regular map with their current location. When one or more SemBeacons are discovered, and their data retrieved, their information is shown on a map. This information may include an indoor map as well as the location of the beacons or devices themselves. Future applications can expand on this to provide contextual actions on the beacons or devices or to visualise information on the map or when interacting with the marker. Our application uses the POSO ontology to extract the positioning system used within the environment. In our dataset, the positioning is limited to multilateration between the fixed locations of iBeacons and SemBeacons.

https://sembeacon.org/examples/openhps2021/beacons.ttl#BEACON_08

```
:pl9_3 a ssn:Deployment ;
  rdfs:label "PL9.3"@en ;
  sembeacon:namespaceId "77f340dbac0d..."^^xsd:hexBinary .

:BEACON_08 a sembeacon:SemBeacon ;
  sembeacon:namespace :pl9_3 ;
  sembeacon:instanceId "c187d748"^^xsd:hexBinary ;
  poso:hasPosition [ a ogc:Geometry ;
    ogc:asWKT "POINT Z(...)"^^ogc:wktLiteral ] .

:BEACON_07 a sembeacon:SemBeacon ;
  sembeacon:namespace :pl9_3 ;
  sembeacon:instanceId "00cc38e7"^^xsd:hexBinary ;
  poso:hasPosition [ a ogc:Geometry ;
    ogc:asWKT "POINT Z(...)"^^ogc:wktLiteral ] .

:BEACON_02 a posoc:iBeacon ;
  sembeacon:namespace :pl9_3 ;
  posoc:proximityUUID "77f340dbac0d..."^^xsd:hexBinary ;
  posoc:major 50174 ; posoc:minor 64267 ;
  poso:hasPosition [ a ogc:Geometry ;
    ogc:asWKT "POINT Z(...)"^^ogc:wktLiteral ] .
```

Listing 6.7: Dataset transformed to POSO

Let us now provide a step-by-step example based on the states described Section 5.2.5. In Figure 6.17, we illustrate our example scenario using an existing dataset [35] with BEACON_7 and BEACON_8 transformed to SemBeacons near the entrance. As soon as a phone arrives in the building, it will passively pick up beacon advertisements from the nearby SemBeacons as well as other beacons advertising on the floor shown as the advertisement indicator (ADV_IND) arrows in grey. Without prior knowledge of the namespace identifier, these advertisements are ignored. Once an AltBeacon-compatible advertisement with an unknown namespace identifier is detected, the phone sends a scan request which triggers SemBeacons to respond with an encoded short resource URI.

After retrieving the short resource URI, the application performs a GET request which in turn returns the resource with the description of the beacon, metadata about the environment and other beacons stored in this resource. On the left-hand side of Figure 6.17, we show our application visualising the GeoJSON floor layout, the beacons in range and known beacons that are not in range (semi-transparent markers). On the right-hand side, we see the semantic resource describing the SemBeacons as well as any other beacons or sensors in the namespace. In our

scenario, we have two SemBeacons each with a unique instance identifier and a position. Other beacons within the same deployment are identified as iBeacons.

```
PREFIX sembeacon: <http://purl.org/sembeacon/>
PREFIX poso: <http://purl.org/poso/>
SELECT ?beacon {
    ?beacon a poso:BluetoothBeacon .
    { ?beacon sembeacon:namespaceId "..."^^xsd:hexBinary }
    UNION {
        ?beacon sembeacon:namespace ?namespace .
        ?namespace sembeacon:namespaceId "..."^^xsd:hexBinary
    } .
}
```

Listing 6.8: Example SPARQL query to retrieve all beacons belonging to the same namespace

In Listing 6.8, we showcase a simple SPARQL query to demonstrate how our application retrieves all beacons belonging to the same namespace. This can either be a SemBeacon with the `:namespaceId` predicate or a deployment that in turn has a `:namespaceId`.

6.7 Discoverable and Interoperable AR

In Section 6.6 we used SemBeacon to discover Internet of Things (IoT) devices and provide contextual information to users. To validate the discoverability of positioning systems as stated in research question RQ3 and the validation of the genericness of the POSO ontology introduced in Section 4.2, we created the fiducial marker ontology named FidMark [41].

FidMark served as a validation for the POSO ontology to be extended for use cases beyond what was originally envisioned. The ontology is detailed in Appendix B.2.3. As part of the validation of this ontology, we developed a prototype for tracking fiducial markers and positioning virtual objects relative to these markers. Our prototype was built on top of OpenHPS. We created a web application that provides custom nodes for (1) detecting ArUco [243] fiducial markers using js-aruco2[20] and (2) a sink node for displaying virtual objects in the scene using Three.js.

Figure 6.18 showcases the demonstrator web application of the FidMark ontology[21] with a fiducial marker placed on a movable object and a virtual object positioned relative to this object. The AR application works in real time using the

[20] https://github.com/damianofalcioni/js-aruco2
[21] https://fidmark.openhps.org/application/

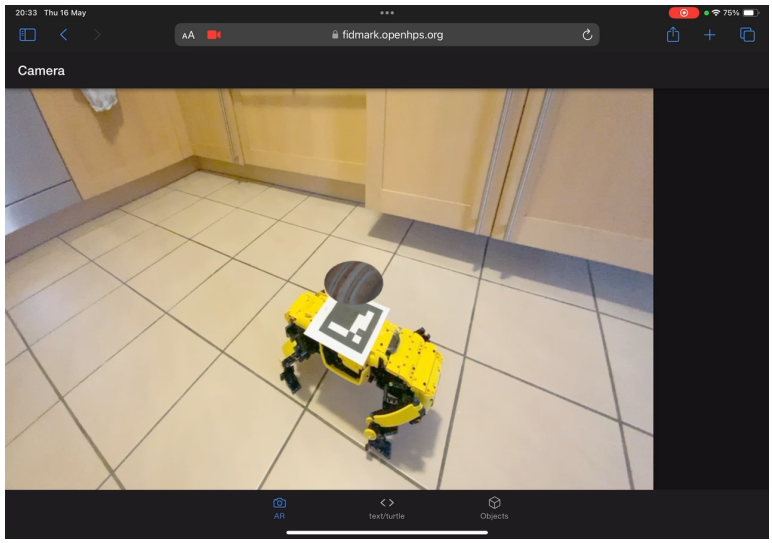


Figure 6.18: Fiducial marker with a relatively positioned virtual object

OpenHPS framework, and it was our prototype that uses AR in our framework, while also leveraging the ability for OpenHPS to run in a browser.

The graph for the AR example is trivial, starting from a video source node and following a single processing lane that computes the *pose* of the marker and visualises the virtual objects. Based on some of the feedback we received for storing positioning data in Solid through properties [43], we evaluated a new technique that enables the storage of virtual environments in Solid Pods.

Our proposed solution aims to allow multiple AR devices to contribute to a single shared AR environment or virtual space belonging to a user. We assume that the AR device used to contribute to augmented environments is a smart device with access to the Web and can broadcast an RF signal. In the general architecture of our proposed solution, we let AR devices broadcast a semantic Bluetooth Low Energy (BLE) beacon advertisement containing the URI of a specific resource. We utilise the SemBeacon advertising protocol that advertises an AltBeacon advertisement and Eddystone-URL-compatible scan response to broadcast semantic data URIs. The use of SemBeacon to advertise the environment URI is illustrated in Figure 6.19. This environment resource contains information about the personal environment(s) owned by the user. Other devices can receive these advertisements when in proximity to the AR device and then access the URI to retrieve more information. For each environment, we have a link to a public inbox that other users can use to link their own modifications to the environment. Any modifications made to the superimposed space are stored in a Solid Pod owned by the user who made the modifications, which enables users to both contribute to the same

environment as well as control the access rights of modifications made to these environments.

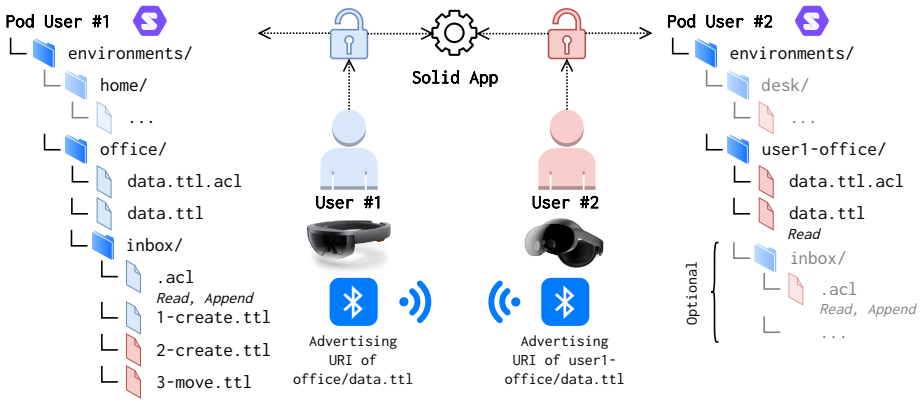


Figure 6.19: A user's discoverable AR environments with two example users (User #1 and User #2) having a Solid Pod

On the left-hand side of the architecture shown in Figure 6.19, we have a Solid Pod for user 1. The Pod contains all environments owned or modified by the user. An AR device connects to the user's Solid Pod through a Solid application that authenticates the user, allowing it to modify the resources when editing a virtual environment. To enable the discovery of these virtual spaces, the AR device broadcasts the `*.ttl` file of the environment it is currently in via BLE advertisements. This resource contains all the information about the environment, such as its location, any identifiable features and all virtual objects placed relative to the environment. While the broadcasted URI is public, the access to read this URI is controllable through access control lists implemented in Solid. SemBeacon offers a broadcasted flag to indicate whether a URI is publicly accessible to prevent applications that do not have access to the resource from attempting to access it.

When another user (e.g., user 2) wants to modify the environment of user 1, they create a new resource including the modifications and additions to virtual objects or detectable features (e.g., markers). The application will then notify user 1 about these changes by referencing the `user1-office/data.ttl` file in the *inbox* [218] container of the environment that is being modified.

An alternative architecture is illustrated in Figure 6.20. In this scenario, a fixed Bluetooth beacon is placed in a room, broadcasting the URI of a single environment. This scenario can be used for public physical environments, such as a meeting room or laboratory, enabling collaboration in AR. Similar to personal environments shown in Figure 6.19, users store their changes to an environment in their Solid Pod and reference these changes in the inbox of the environment.

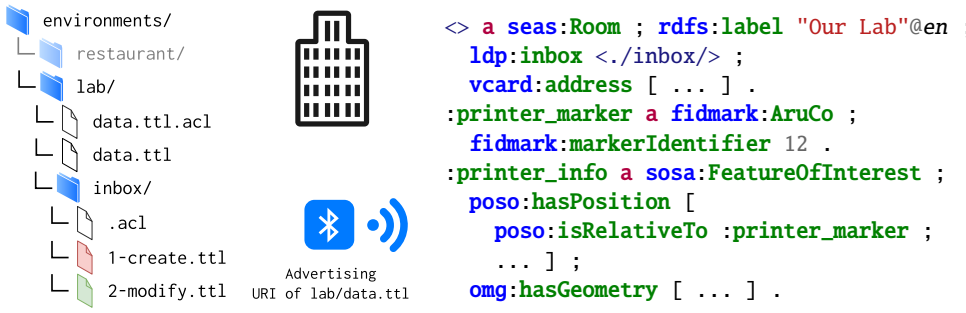


Figure 6.20: Single discoverable AR environment using the `seas`[a], `fidmark`[b], `poso`[c] and `omg`[d] vocabularies

[a]Smart Energy Aware Systems Ontology: `https://w3id.org/seas/`
[b]Fiducial Marker Ontology: `http://purl.org/fidmark/`
[c]Positioning System Ontology: `http://purl.org/poso/`
[d]Ontology for Managing Geometry: `https://w3id.org/omg#`

6.7.1 Usage

Our solution is depicted in Figure 6.21 where we showcase the flow of our architecture previously illustrated in Figure 6.19. Two users with AR devices have their own Solid Pod. User 1 will create an environment (A) on their Pod and subscribe to the inbox container of this environment. Once the environment is ready, the AR device will use the SemBeacon specification to advertise the URI of the environment and enable its discovery.

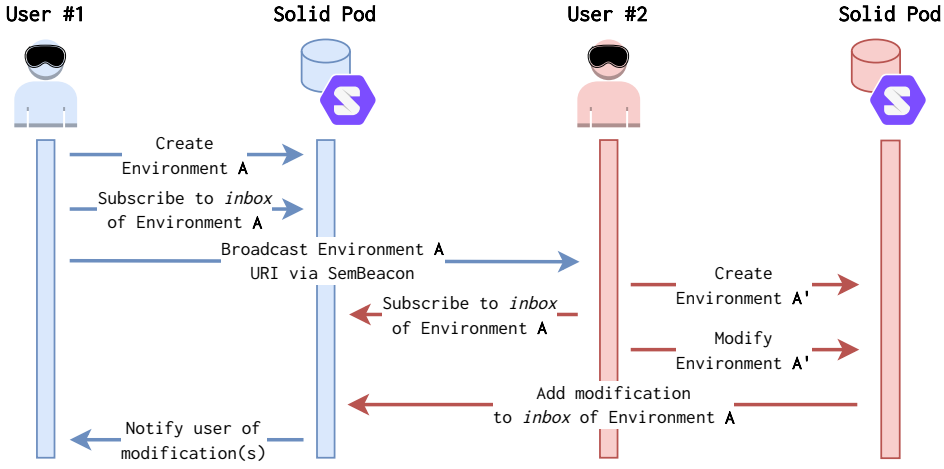


Figure 6.21: Interaction flow of two users contributing to the same augmented reality environment

When another user (e.g., user 2) discovers the resource URI, the AR application will access the environment to visualise the augmented objects. If user 2 makes a modification such as adding virtual objects, these modifications are stored in the Pod of user 2 as environment A', ensuring ownership of this contribution and enabling user 2 to choose the access rights to this modification. To keep up-to-date with changes in the original environment, the application will listen for changes in the inbox of environment A. All users who are subscribed to this inbox will receive linked data notifications [218] whenever a new modification is added. Unlike the i-VISION [244] project which uses RESTful polling to detect changes in the publish and subscribe gateway, linked data notifications can enable real-time updates without the additional data overhead introduced in polling.

An inbox uses the LDP[22] vocabulary to index all resources within this container. A user who wishes to accept contributors for their environment should configure this inbox container with public *append* access rights, allowing other users to append a new resource to the container. Each inbox item represents an action using the schema.org vocabulary [204]. An action represents an event that occurred in the environment, such as creating a new virtual object, moving an object or interacting with a virtual object.

```
@prefix card: <https://user2.../profile/card#>.
@prefix office: <https://user2.../environments/user1-office/data.ttl#>.

<> a schema:CreateAction ;
  schema:description "Created a new object 'painting'"@en ;
  # AR application that created the action
  schema:agent <https://ar-app.com/id> ;
  schema:creator card:me ; # Owner of the modification
  schema:object office:painting ; schema:result office:painting .
```

Listing 6.9: Inbox item to identify the creation of a virtual object

Listing 6.9 demonstrates an individual appended inbox item to the Solid Pod of user 1. In the example, we see a `schema:CreateAction` indicating that user 2 created a new *painting* object. Users can listen for notifications in the inbox to automatically apply changes to virtual objects in the shared AR environment as they are made [218].

The positioning of virtual objects is done using the POSO ontology [40], which allows the description of (virtual) objects to be placed relative to other objects using the `poso:isRelativeTo` predicate. When a user wants to place an object in an environment using an absolute position that is not relative to a marker or detectable feature, the same predicate can be used to indicate that the absolute position is relative to the environment.

[22] `https://www.w3.org/ns/ldp#`

The final AR environment, combining all modifications made by other users, is created by the AR application that has access to the Solid Pod. Our architecture allows users to easily ignore all contributions of another user or agent. Individual modifications or contributions can be rejected by ignoring the individual inbox items where these changes are referenced. Future work might address the moderation of individual contributions by applying quality assessment crowdsourcing techniques [245].

6.7.2 Reference Frame

Augmented reality uses one or more reference frames [246] to anchor virtual objects in the physical environment and to determine an absolute or relative position. This can be done through feature detection [247] that creates anchors based on visual patterns or artificial features such as fiducial markers [248].

To enable interoperable applications to contribute to the same environment, all applications need to operate within the same reference frame. Our solution assumes that contributions to an environment not only include virtual objects, but also additional anchor points such as markers and detectable features contributed by multiple users [41]. By semantically describing these reference frames as visual landmarks rather than a frame of reference, the POSO ontology can be used to position virtual objects relative to these anchors.

6.8 Discussion

In this chapter, we demonstrated several examples of (indoor) positioning systems and discussed various applications and evaluations of these systems. We showcased prototypes such as the user-centric storage in Solid Pods and our SemBeacon application. All the prototypes shown in this chapter were developed using the OpenHPS framework. We also used these prototypes to further improve the framework and its data representation, and to pave the path towards an integrated solution.

By building and testing each of our prototypes using OpenHPS, we provide future researchers with reusable components and algorithms to conduct further experiments. Furthermore, it also provides researchers with usable artefacts for using ontologies such as POSO and FidMark in real-world applications. Applications, prototypes and experiments in this chapter were conducted as technical evaluations of the frameworks, ontologies and specifications presented in this dissertation. Furthermore, by conducting these technical evaluations, we have revealed several limitations which we were able to address in the current version of our OpenHPS framework and the integrated solution.

Walking on water and developing software from a specification are easy if both are frozen.

– Edward V. Berard

Chapter 7

Integrated Solution

Based on the answers to our research questions provided in Sections 3, 4.2, 5.2 and 5.3, we designed an integrated proof of concept solution enabling the interoperability and discoverability of positioning systems. With our integrated solution, we propose an architecture for developing new positioning systems that offer interoperability of the data they produce, as well as consumer applications, such as navigation applications that require location data to function.

We start with a user analysis on the stakeholders of our integrated solution. Next, we list the requirements of our proposed solution in Section 7.2. Using these requirements and our solution for defining the data in hybrid positioning systems (RQ1), we define the actors of our solution in Section 7.3.

The final integrated solution extends on our previous work with the SemBeacon application, which already implements OpenHPS, POSO, the scanning of semantic beacons and the visualisation of environments. Together with our work with Solid, we designed an architectural solution that offers user-centric storage of data and uses Solid as a distributed registry for positioning systems using our Linked Data Hash Table (LDHT) solution.

With Objective 5, we wanted to determine the feasibility of our solution for interoperable and discoverable indoor positioning systems. We detail the architecture of the solution and illustrate how our individual contributions fit within this architecture in Section 7.5. While our implementation uses Solid, the conceptual design of our integrated solution is generic and can be implemented using different personal data vaults.

By separating our architecture into different components, we enable developers to adapt and extend the solution according to their needs. Furthermore, the use of Solid and Linked Data principles ensures data privacy, user-centricity, and control over personal data.

7.1 User Analysis

Before determining the requirements of our integrated solution, we must first identify the various users involved in the project. The primary stakeholders in this case are the end users who will be using the integrated solution on a day-to-day basis.

Next, the building owners who wish to deploy such a system are also important. Without their support, the deployment of an indoor positioning system would not be possible. Finally, the developers and technicians responsible for implementing, deploying and maintaining an indoor positioning system should also be considered. For the scope of this analysis, we consider the industry players that develop these systems as part of their business goal to be part of this group. These three main stakeholders, along with the possible advantages our integrated solution would bring, are further detailed in this section.

7.1.1 End Users

We consider end users to be those who use an indoor positioning system on a daily basis. Use cases can vary from navigation to the tracking of assets within a building, meaning they may not be the actual users or objects being tracked by such a system.

For users who are tracked, our solution offers the ability to store the data obtained by such a system in a user-centric storage. This provides them with control and transparency on the applications and services that have access to read and/or write to their managed storage.

Interoperability among indoor positioning systems opens the doors for reusable applications and other interfaces. This allows users to choose which application(s) to use rather than being forced to use the application associated with a particular building. Furthermore, in current SOTA, users would have to switch between applications when navigating from one building to another. With reusable and interoperable applications, this need would be prevented by offering seamless positioning.

7.1.2 Building Owners

As detailed in Section 2.2, deploying indoor positioning systems is costly due to the need to deploy hardware, calibrate environments and develop software [78]. With our integrated solution, we aim to pave the way for interoperable systems, while also encouraging more building owners to deploy such a system.

We believe that by providing reusable applications and components, we can limit part of this cost. Additionally, by enabling interoperability among these systems, a

building owner can easily reuse a deployed system and its associated data for other use cases. By also facilitating the discovery of deployed systems, building owners who want end users to benefit from an IPS can be targeted more effectively.

7.1.3 Developers and Technicians

We want to offer developers an architecture that they can use to design an (indoor) positioning system, capable of integrating with existing applications and other systems. Instead of forcing developers to start from scratch, we want to enable the creation of building blocks and reusable software that allows them to focus on modifications rather than reimplementation.

Configuring positioning systems often requires a setup phase in which technicians collect data within a building. As these applications are often only used once, they are either very generic (e.g., GetSensorData [249]) or lack efforts to enhance the user experience. Reusable applications would entail that more time can be spent on designing these *single-use* applications, as they can be reused in different buildings. This would benefit both the developers who implement these applications, as well as the technicians who are the primary users of such applications.

From a business perspective, the ability to reuse applications limits the need for developers to be hired to develop such software. However, we believe that providing them with reusable building blocks helps advance the domain to better standardisations and interoperability beyond location data. However, we acknowledge that some industry players may see the reusability and accessibility of data as a competitive disadvantage.

7.2 Requirements

In the following, we formalise the requirements of our integrated solution, which combines the answers to all our research questions. The requirements combine the individual requirements of hybrid positioning systems, interoperability of these systems and finally the discovery.

7.2.1 Functional Requirements

We identified the functional requirements by analysing the needs of users, system capabilities, and potential use cases. These requirements include the ability to collect data from different positioning systems, integrate the data into a common format, perform fusion algorithms to improve accuracy, and provide an interface for users to access and manipulate the integrated data.

The requirements for our integrated solution represent high-level requirements for the functioning of our solution. These requirements were established based on

our OpenHPS requirements, as detailed in Section 3.2, as well as the requirements derived from our research questions.

Track Persons and Objects

An interoperable positioning system should track persons or objects in a specified environment. This means that multiple positioning systems that can *behave* in an interoperable manner should be able to track the same individual object or person. This functional requirement also indicates that these individuals should be identifiable between systems.

Processing of Data

Multiple applications or systems should be able to process the data produced by an interoperable positioning system as detailed in Definition 5. Data such as a computed absolute position, orientation or relative position should be integrated and processed seamlessly, allowing for the utilisation of the data in various applications or services.

Access Control and Privacy

Individuals who are tracked should be able to control which positioning systems and applications have access to their data. This functional requirement ensures that we can enable access to data while also ensuring that the privacy of sensitive location data is retained. In addition to this requirement, we include the ability for users to provide consent to the systems and services that produce and consume location data. In the case of objects that are being tracked by a positioning system, the owner of the object should have control over which systems and services can access the data.

Discovering Entities and Systems

Positioning systems should be able to discover and identify entities that can be tracked. Vice versa, an entity should also be able to discover if a positioning system exists within a nearby environment. This functional requirement should still adhere to other requirements such as the access control and privacy rights of users. While we want systems to discover entities, this should happen privately with the consent of the user.

7.2.2 Environment Requirements

These requirements include factors such as the type of infrastructure available, the area coverage needed, and the level of accuracy required for positioning data.

By considering both functional and non-functional requirements, along with environmental factors, our integrated solution is designed to effectively address the challenges of interoperability and discoverability in positioning systems.

Physical Environment

The architecture is to be deployed in a physical environment and does not apply to virtual positioning systems that operate in a fictional environment such as a virtual world. However, virtual positioning systems can still be added on top of our architecture under the condition that the environment of the system mimics a physical environment.

This environmental requirement scopes our architecture to the physical world, rather than designing interoperable positioning systems that could also be used to ensure seamless data exchange in video games. While open use cases exist to ensure interoperable exchange of location data in video games [250], these would also require interoperable virtual reference frames, which we do not tackle in this integrated solution.

Positioning Infrastructure

Positioning systems can use a wide range of infrastructure, such as beacons or visual markers. An interoperable solution should be aware of this available infrastructure and the requirement for this infrastructure to be documented.

7.2.3 Data Requirements

Most of the data requirements for positioning systems have already been defined in Chapter 3. In this section, we focus on the data requirements for our integrated solution, with the additional requirement of making the data interoperable and ensuring privacy and transparency for the user.

Generic Input and Output

When creating an integrated solution, the data that is stored by this solution should offer a generic input and output representation of positioning systems.

User Data Consent

User data, such as location data, is private to the user. Our integrated solution requires a user's consent to ensure that users have control over who can access their location data. This requirement aligns with the access control and privacy rights of individuals, allowing them to provide consent for the systems and services that collect and use their data.

Interoperable Data

Data generated and used by a positioning system should be semantically described in order to support interoperability between applications and systems. This includes the input and output data of a positioning system.

Furthermore, with our integrated solution, we adhere to the *Open World* assumption, which states that we cannot rely on the fact that data does not exist if it is not available within a knowledge base. The same applies to the location data of users. Simply because our integrated solution has no knowledge of services and systems that track a subject, we cannot guarantee that all location data is accounted for.

Interoperable Environments

An environment where a positioning system is deployed should be semantically described so it can be recognised by other systems. This includes information about the physical layout, boundaries, obstacles, and other relevant features of the environment. By meeting these data requirements, our integrated solution ensures that positioning data can be easily shared, combined, and used across different systems and applications.

7.3 Actors

In Chapter 3, we already defined different actors in a hybrid positioning system. We defined a *tracked actor*, *tracking actor*, *calibration actor* and a *computing actor* as the four actors within OpenHPS. In our integrated solution, we also want to consider consumers of the output data from a positioning system, as well as the storage in which this data is located.

We therefore generalised these actors even further by classifying the individual parts of a positioning system as a data producer, data consumer, data processor and data store. This allows us to consider position visualisation applications that do not provide any tracking as a data consumer. Similar to the OpenHPS actors, the object or person that is being tracked will be called the *tracked subject*. Individual sensors that are part of a positioning system are also data producers in their own scope of data. The calibration actor, which was present in OpenHPS is considered a data producer that stores calibrated information in a certain storage. Each actor is represented in Figure 7.1.

Tracked Subject

The tracked subject of the architecture is the object that is subjected to the tracking of a positioning system. This tracked subject can be a person, robot or any other asset that must be located. Examples of tracked subjects include mobile robots,

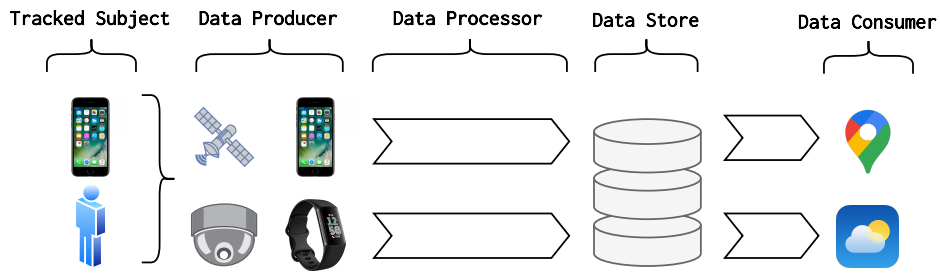


Figure 7.1: Actors of positioning systems

vehicles, and individuals in a smart environment. The tracked subject interacts with the positioning system by generating data that can be used for positioning and tracking purposes. Additionally, the tracked subject may also consume information from the positioning system, such as receiving alerts or notifications based on their location.

Data Producer

Similar to other stream-based processing systems such as Apache Kafka [179], a data producer in our architecture is any third-party service or system that creates or updates information about a tracked subject. This data can include real-time location updates, sensor data, or any other relevant information that can be used for positioning purposes. Data producers can range from IoT devices and mobile applications to external positioning systems.

Furthermore, the data producer is responsible for representing data in a standardised format that adheres to the semantic description of the data requirements. This ensures that the data can be easily consumed and utilised by the positioning system and other relevant actors within the ecosystem.

Data Processor

Data processors are any algorithmic procedure that manipulates produced data. A data processor can be a single procedure or a pipeline of procedures that processes the same data. This actor describes a positioning system and a data producer that processes sensor data.

Data Store

Finally, the last actor that can be found in all positioning systems is the data storage actor. Data producers produce data that is temporarily stored or persisted in a data store. This can include simple data storage, such as a temporary in-memory data store to more intricate cloud-based storage of information.

Data Consumer

A data consumer in our architecture is any third-party service or system that *consumes* information of a tracked subject. This includes applications that visualise the position of tracked subjects as well as analytics tools that process the data generated by the positioning system. Data consumers can range from navigation applications to data analysis platforms. Furthermore, in the context of hybrid- or interoperable positioning systems, a data consumer actor also applies to the logic that consumes information from another service to aid in the tracking of a tracked subject.

7.4 Use Cases

The integrated solution we propose in this dissertation is based on the problem statement outlined in our research questions and the varying use cases in which an (indoor) positioning system is used. A general overview of the different domains where indoor positioning systems are used is provided in Section 2.2.1. In this section, we delve deeper into specific use cases where our interoperable and discoverable integrated solution innovates.

7.4.1 Navigation System

A navigation system is a positioning system that aims to guide a person to a specific destination using visualisation techniques. As opposed to navigation systems for robots which aim to guide autonomous vehicles to navigate to a location by following a path, we assume that the tracked subject in this use case are humans who need to follow textual or visual instructions. Outdoors, these navigation systems use a single technology (i.e., GPS). To create an interoperable navigation system, the system should perform seamless positioning with a handover from one positioning system (i.e., outdoors) to another (i.e., a building-specific indoor positioning system).

In previous work, we developed an indoor navigation system that makes use of simplified instructions and visual aids to direct the user to their destination [99]. If we consider the use of a single *navigation application* that works both indoors and outdoors, we require a method that uses GPS outdoors while seamlessly transitioning to other positioning techniques when entering a building.

7.4.2 Indoor Positioning System

Indoor positioning systems are systems that track a person or object within an indoor environment. Outdoors, we can rely on public maps to provide context to an outdoor environment. However, buildings and internal floor plans are often not

considered public. Even if a building is public, there is often no method to retrieve indoor mapping information. Our integrated solution would allow the discovery of these indoor positioning systems and the environments (e.g., floor plans or digital representations) in which they are deployed.

7.4.3 Asset Tracking

Asset tracking is a service that tracks assets such as cars, equipment or inventory. In this scenario, we consider assets as non-users who cannot consent to the tracking. However, assets are usually owned by an organisation or user. Our integrated solution should provide users with a solution to add multiple assets that can be tracked individually.

7.4.4 Collaborative Robots

Collaborative robots, also known as *cobots*, are robots designed to work alongside humans in a shared workspace. In an industrial setting, these robots can assist human workers with tasks that may be repetitive, dangerous, or physically demanding. For effective collaboration between humans and robots, precise positioning and tracking of both entities are crucial. Our integrated solution can provide real-time monitoring and communication between collaborative robots and human workers within a shared workspace.

7.4.5 Augmented Reality

Augmented Reality (AR) is a technology that superimposes digital information onto the real world, typically viewed through a smartphone or AR headset. In most modern implementations of AR applications and devices, each AR device is responsible for its own tracking and frame of reference [246]. One use case for our integrated solution is to enable interoperability between multiple AR devices and to enable AR devices to discover each other when in the same environment. This type of interoperability will allow for collaborative AR experiences where multiple users can interact with the same digital content in real time. Early work on this type of interoperability was developed in a prototype in Section 6.7.

7.5 Architecture

Our solution relies on the use of personal data vaults for every person and object that can be tracked by a positioning system. In our architecture, the tracked subject is central to the collection, accessibility and interoperability of the data. For our solution, we make use of the Solid project detailed in Section 2.5.2. Solid provides each user with a personal data vault that can store both binary and linked data.

Positioning systems are represented as graph-based processing networks that take input data and output a position along with possibly other information about the tracked subject. This graph-based representation is further detailed in our OpenHPS framework, which generalises the structure of most hybrid positioning systems. Each step in this process network is described as a procedure. When we store data, we also keep track of the procedure(s) used to process this data. This gives us knowledge on how certain data is manipulated, providing applications with more understanding of the relevancy and accuracy of this data.

For expressing positioning systems as a graph structure, we make use of our POSO ontology. POSO provides a semantic description of the individual procedures within a system. In addition, POSO also provides a vocabulary for describing the data produced and consumed by such systems.

Finally, the discoverability of the tracked subjects and positioning systems alike will be done using physical media. Whenever we are talking about a positioning system, we represent it as something in a *physical space* that aims to track *physical objects*. To aid in this discovery, we developed the SemBeacon specification [45] which uses Bluetooth Low Energy (BLE) to transmit the URI of the semantic resource. However, other physical mediums such as a simple Quick Response (QR) code can be used to discover the positioning system at a particular location.

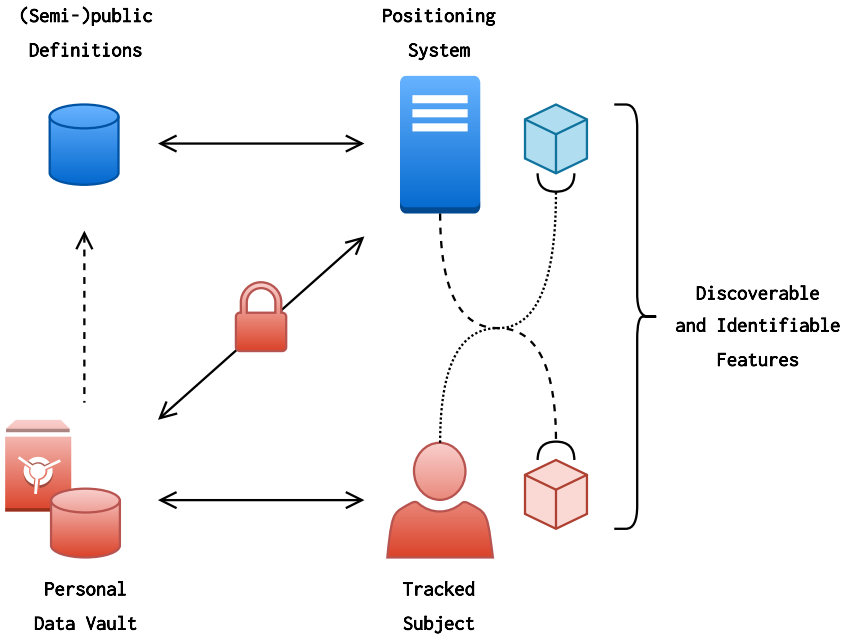


Figure 7.2: Architecture of an interoperable positioning system using personal data vaults and discoverable features

The basic setup of our architecture is illustrated in Figure 7.2. In blue, we have the positioning system along with its semantic definition. This definition can be completely public or private, as long as the positioning system can be discovered via a discoverable and identifiable feature. This feature can be anything that can be mapped or resolved to the URI of the semantic definitions. The tracked subject and its data vault are indicated in red. A tracked subject must provide the positioning system with access to its personal data vault. Depending on the granted access rights, the positioning system can read, update or create new information about the tracked subject in its own personal data vault. Other positioning systems with similar access rights will be able to continue the tracking using this same personal data vault. Positioning systems can also decide not to reveal all internal algorithms and procedures publicly, similar to a location-based service (LBS). In this case, the information is semi-public since it still contains a minimum amount of information needed to discover the system.

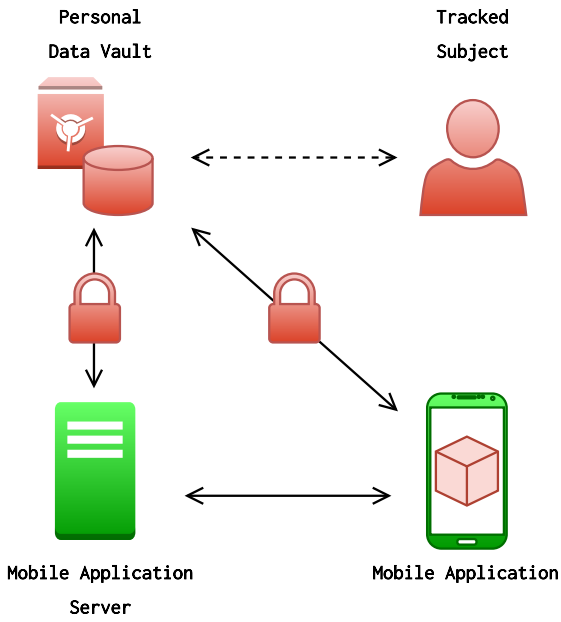


Figure 7.3: Architecture of an interoperable positioning application using personal data vaults

In Figure 7.3; we consider a third-party mobile application and a third-party server (illustrated in green). The mobile application becomes the discoverable and identifiable feature of the tracked subject. Both the application and server require access rights to the personal data vault of the tracked subject. Optionally, only the server can request access rights, in which case the mobile application interfaces with the personal data vault via its own authenticated server.

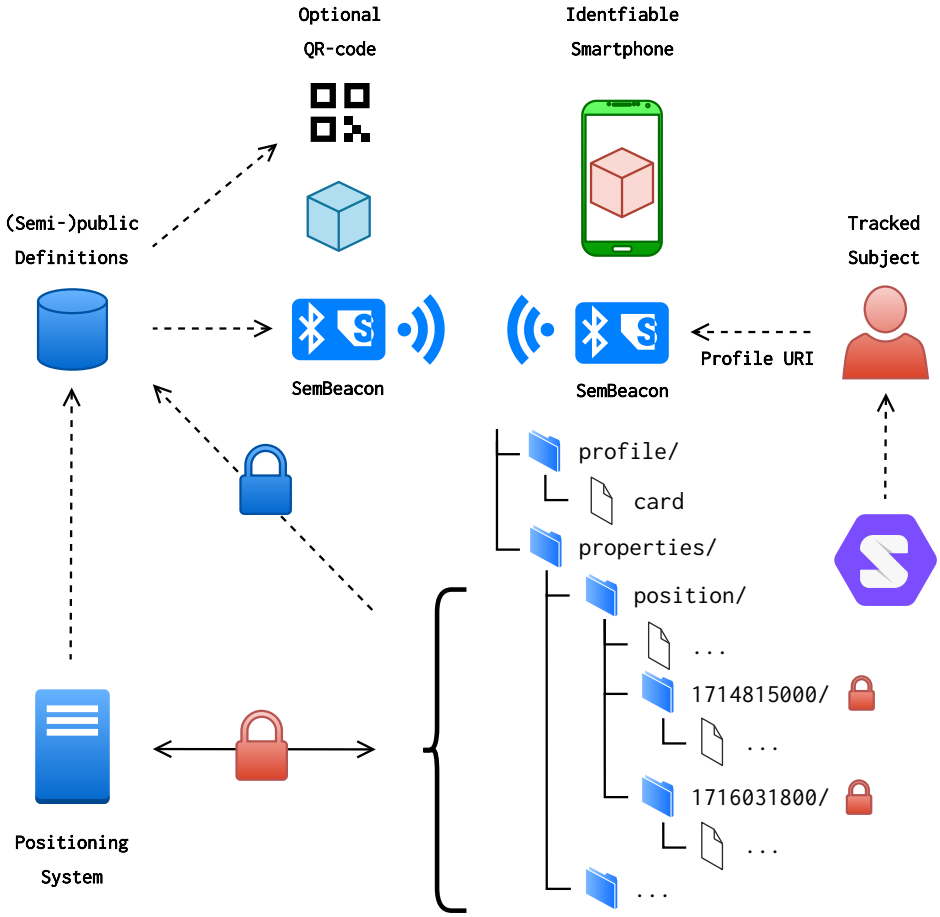


Figure 7.4: Architecture of our integrated solution including Solid and SemBeacon

The complete architecture of our integrated solution is shown in Figure 7.4. It consists of a tracked subject that manages a Solid Pod, a positioning system that has a semantic description of its algorithms and technologies. This description can be public or private, depending on the use case of the positioning system.

Both the tracked subject and positioning system use the SemBeacon specification to act as their *discoverable feature*. For a tracked subject, the profile URI is advertised as the *identifiable feature*, while for positioning systems it is the URI to a description of the `poso:PositioningSystem`. While our integrated solution proposes the use of SemBeacon to act as the discoverable feature, other techniques such as QR codes which act similarly can be used.

A user's Solid Pod contains the profile card that is advertised using the SemBeacon specification. In addition to this profile, a container with properties is kept that structures different observable properties (e.g., position, orientation or velocity)

into Linked Data Event Streams (LDES). Users can provide positioning systems with access to their properties, enabling them to add new observations of these properties or to make use of this data to further improve positioning accuracy.

The positioning system used within our architecture can include a wide range of techniques and algorithms. As long as they can be semantically expressed, they can be integrated into our framework. With OpenHPS, we have already validated that various systems and setups can be configured as a graph structure. In our paper, *Indoor Positioning Using the OpenHPS Framework* [33], we demonstrated how we could use Wi-Fi fingerprinting, Bluetooth Low Energy beacons, and dead reckoning techniques to determine a position. Unlike our previous paper, positioning systems designed with our integrated solution would not require a proprietary application for each building that deploys an indoor positioning system.

7.6 Linked Data Vocabulary

To enable interoperable input and output data of a positioning system, as well as an interoperable description of the system itself, we require a linked data vocabulary. Our vocabulary mainly relies on the POSO ontology detailed in Section 4.2 along with additional extensions. While the vocabularies listed in this section offer a baseline that we would use within an integrated solution, linked data can be expanded to any domain, any use case and any open problem.

7.6.1 Positioning Data Vocabulary

Any data that represents computed positioning data (e.g., a position, orientation, velocity) should be representable using a linked data vocabulary. Each system may provide data with a certain accuracy and unit.

In Section 4.2, we introduced our Positioning System Ontology (POSO) aimed at a wide range of positioning solutions, which enables us to semantically describe individual *observations* of *properties*. This ontology generically represents location data with the assumption that positioning systems do not always use a geographical coordinate reference system.

In the environmental requirements of our integrated solution, we scope the solution to physical indoor environments which are bound to geographical coordinates. This allows us to make the assumption in our solution that we are working with geospatial data.

7.6.2 Sensor Data Vocabulary

Positioning systems generate positioning data that is computed from raw sensor data. Part of the definition of interoperability is the ability to access data, including

any data that is used by the system to process information. In our integrated solution, we propose the use of the Machine-to-Machine Measurement (M3) lite ontology [203], which enables the description of sensors. We have aligned this ontology with POSO using the `poso-m3lite` ontology alignment. For the scope of our integrated solution, we focused on the interoperability between positioning systems with the data they produce.

7.6.3 Positioning System Vocabulary

Being able to represent the sensor and computed positioning data already enables the interoperability of different positioning systems. However, to reason on the importance of the computed positioning data, we also need to describe the positioning system itself using a linked data vocabulary. This includes the capabilities, constraints, and requirements of the system to enable seamless integration and communication between different positioning systems. By defining a standardised vocabulary for positioning systems, we can ensure that data sharing and interoperability are achieved at a high level of accuracy and efficiency.

Similarly to the positioning data vocabulary, we use the POSO ontology to describe the individual algorithms and technologies these systems employ by defining them as *procedures* within a pipeline. We designed an object-document mapping tool for TypeScript to enrich classes with metadata which provides information about their semantic definition. On the OpenHPS website, we provide a URI for every single processing node that is created by an official module, enabling positioning systems created with OpenHPS to explicitly indicate which implementation was used rather than providing a general description of the algorithm.

7.6.4 Access Control Rights and Discovery Vocabulary

In our proposed integrated solution, we aim to allow the user to control access to their location data. Multiple standardisations already exist to enable access control rights, such as the Access-Control List (ACL) [251] or Web Access Control (WAC) [252].

Before a positioning system can access resources in a personal data vault, the user needs to authenticate themselves with the system. The authentication flow of the application is detailed in Section 2.5.2, but in general, it requires user interaction for authentication.

7.6.5 Authentication and Consent

The public definition of a positioning system should provide information for users on how to authenticate their personal data vault with the system. However, in

addition, it should also provide contextual information on how the system plans to use the data.

The Data Privacy Vocabulary [253] provides a comprehensive vocabulary for describing policies, consent and the basis for processing of data. Using this vocabulary, a positioning system can detail to users how they track a user's location or utilise data from their personal data vault before they authenticate.

7.7 Personal Data Vault

To provide users and organisations with a personal data vault that they can use to store their data, we propose to make use of the Solid project (see Section 2.5.2). Solid allows users to create personal data vaults, often referred to as a *Pod* that they register at a *Pod* or *Storage Provider*. They can choose which provider they use and under what terms and conditions the storage provider offers to store their data. These data stores can store binary data and linked data that pertains to the user who owns the storage.

An advantage of Solid is its ability to enable access control to third-party data providers and consumers who want to access (part of) the data within their personal data vault. In the scope of our integrated solution, this means we can enable users to decide which location data is shared with which application or service.

Each Solid storage has a public user profile linked data resource (LDR) containing user information such as basic personal information. We expand this profile with a set of observable properties that this person can have using the *ssn:Property* predicate described in Section 4.2. This allows applications that read a user's profile to know which *properties!observable* can be found in the personal data vault.

7.7.1 Data Structure

The data that we store in the personal data vault will be structured to support Linked Data Event Streams (LDES) [213] while maintaining individual streams for location, orientation and velocity data. This structure is necessary to enable us to store a large stream of data. Storing a stream of data in a single resource as we proposed in our original work with Solid [43] is not feasible due to scalability issues.

Figure 7.5 showcases the structure of properties such as a *position* or *orientation*. Each property is located in its own LDP container and contains all the data concerning the property, including its description or individual observations, regardless of the source or accuracy of the data. To structure the data in a way that allows multiple data producers to write new data without causing a single resource to

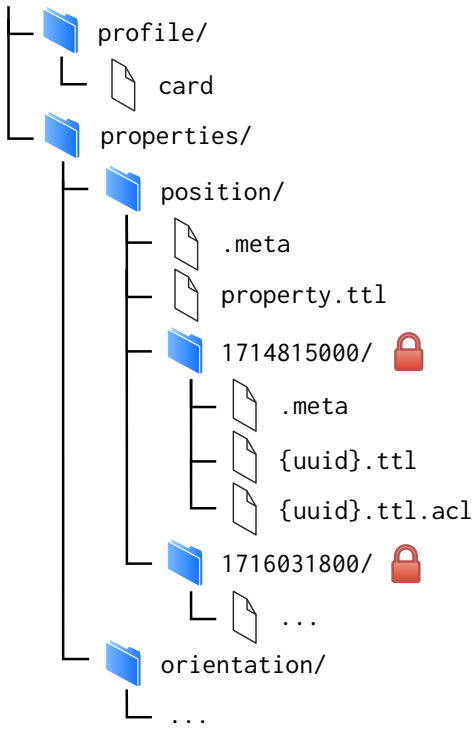


Figure 7.5: Container structure of properties

become unmanageable, we use a TREE hypermedia structure [216] with a single event stream for each property. More information about the TREE hypermedia structure and LDES is provided in Section 4.4.3.

Data of the property is structured into multiple containers that contain a subset of the data and act as a node within the TREE structure. Each data point (i.e., observation) at a particular timestamp is stored as a separate resource (i.e., `.ttl` file). The root node, located in the `property.ttl` file provides information on the distribution of the data and indicates which child nodes contain information for a certain timestamp. Positioning systems should have *append* permissions to the property container and write permission to the `property.ttl` file. Using these permissions, new containers can be created while also ensuring that child nodes are added to the root node in the property resource.

Different levels of granularity can be stored by indicating the accuracy as a specific version using LDES. Regardless of the level of granularity, the data is stored with the same event stream, sorted by the timestamp when it was created. Future work will have to improve this method, as currently, it would require complex techniques to provide access control to a certain granularity of data.

The data structure is stored as a data shape for each property, providing a consistent *blueprint* for other applications and positioning systems. For the scope of our integrated solution, we will not store real-time sensor data such as accelerometer data within personal data vaults. The vocabularies to represent this information exist, such as M3 Lite [203] and our alignment with POSO. Data would be stored similarly to properties within their own containers to enable a solution that can scale with large amounts of streaming data.

7.8 Tracked Subject Discovery

Positioning systems aim to track objects or users within their operating environment. Identifying these tracked subjects within a spatial environment requires a method to discover and identify subjects based on a set of features.

The architecture proposed in our integrated solution envisions the broadcasting of Web Identifiers (WebID) [140] through semantic beacons. Tracked subjects that wish for positioning systems to detect their presence can optionally enable the discovery of their WebID. Semantic beacons, implemented as Bluetooth beacons adhering to the SemBeacon specification introduced in Section 5.2, will broadcast a shortened WebID URI together with a flag that identifies the WebID as a movable object.

Similarly, the discovery of a positioning system can be achieved by advertising the presence of the organisation that deployed the system. Contextually, this knowledge can aid users and applications in determining the relevance of the positioning system.

In Figure 7.6, we demonstrate the SemBeacon application receiving the WebID of a tracked subject. The WebID acts as the identifiable feature of the tracked subject, while the advertising using BLE represents the discoverable feature of the tracked subject. With our integrated solution, the identifiable WebID indirectly links to the *properties* of the tracked subject through the profile document, allowing us to discover the position of a person or object.

7.8.1 User and System Flow

In this section, we will outline the user and system flow for discovering and identifying tracked subjects within the spatial environment. We consider two main groups of scenarios based on the use cases listed in Section 7.4, more specifically (1) scenarios where the user initiates the search of a positioning system and (2) scenarios where a user is tracked without their (prior) consent.

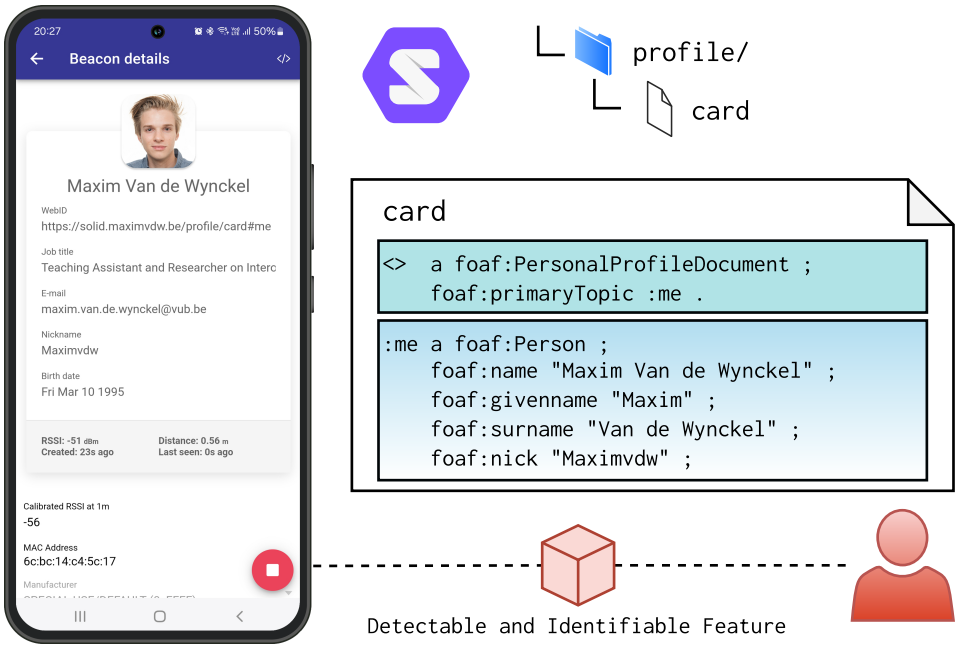


Figure 7.6: WebID advertising of a tracked subject through the SemBeacon application

User-initiated Tracking

User-initiated tracking involves all use cases where a user initiates the tracking by *manually* searching for a nearby positioning system. This manual task can be accomplished by actively searching for new systems or by passively discovering systems that are relevant to the user. While we assume this is a manual task, we also consider scenarios where automated processes can assist in the discovery of positioning systems based on user preferences or historical behaviour. However, due to the data requirement of user intervention to provide permissions to the user data, we assume that the user initiates the tracking.

(I) The first step involves the user using an application or *background service*[1] to scan for BLE signals. This is a step that is continuously performed as long as the user wishes to find positioning systems. At the same time, a BLE signal is broadcast containing the user's WebID. This broadcasting is optional and only needed when the user wishes positioning systems to discover, track and identify the user.

[1]An application that runs in the background and does not require user interaction unless a positioning system is found

(II) Once a BLE signal is discovered that can be matched to a semantic beacon containing a positioning system, the application will access the online description of the system to determine its relevance for the application[2].

(III) The online description of a positioning system that aims to store information about a user requires public information that indicates how users should authenticate themselves with the system and how they can provide the system with access to their Solid storage.

(IV) Once authenticated, the system will check the available properties of the given user. A user can have multiple observable properties, some of which are not relevant to the positioning system. When the positioning system can provide properties that are not yet available in the user's Solid storage, they will be created.

(V) When the relevant properties are found or created, the positioning system can start to add observations of these properties using a linked data event stream structure.

System-initiated Tracking

Existing use cases of positioning systems, such as surveillance, asset tracking and supply chain management involve scenarios where a user is tracked without their consent. In these scenarios, the tracking is initiated by the system itself based on predefined parameters or rules.

For system-initiated tracking, we only consider cases where users will eventually be able to choose to opt-in to the tracking system. This opt-in process is crucial to enable the storage of the data in their personal data vault.

(I) A user is tracked by a non-trivial set of positioning techniques. An internal *profile* is created for the user in the system that contains the output by the system. For example, a positioning system may track a Bluetooth device without knowledge of who it belongs to.

(II) The user enables their discovery by advertising their WebID via the Bluetooth SemBeacon protocol.

(III) The system detects the advertised WebID and links this identifier with the internal profile that contains the location data. At this point, a user is uniquely identified, but data is not stored in a user's personal data vault.

(IV) In the WebID, a reference to the properties (e.g., position) of the user is available. Depending on the permissions, the system can write data to this

[2]Contextual relevance depends on the use case of the application, the proximity of the positioning system and other information

property. In case the system does not have permission to write to the user's Solid storage, a request has to be made to the user to request access.

7.9 Distributed Registry

In Section 5.3 we introduced our Linked Data Hash Table (LDHT) specification. As an implementation of this specification, we created the POSO collection registry, a distributed hash table of positioning systems using LDP containers in Solid.

This collection registry aims to provide global discovery of positioning systems by letting applications globally search for nearby positioning systems based on an approximate location estimate, such as a rough GPS position or user input. We envision that interoperable and discoverable indoor positioning systems are not only available via SemBeacon advertisements but also that their URI is available in the distributed registry. This allows applications to dynamically discover and connect to nearby positioning systems based on user preferences and requirements.

Interoperable positioning systems that implement our architecture could request users if they are willing to let their Solid Pod be added as a peer to the LDHT network. This would ensure that users can help others in the discovery of positioning systems.

7.10 Discussion

Our integrated solution offers an architectural blueprint to design interoperable and discoverable indoor positioning systems. However, the quote by software engineer Edward V Berard, *"Walking on water and developing software from a specification are easy if both are frozen."* is also applicable to our work. While most specifications used within our integrated solution have been tested for several years, newer specifications such as LDHT may still require some changes that influence our presented solution. Moreover, our solution relies on the Solid project, which is a living project that is still under construction. Changes to the Solid specification may also influence the design of our solution or facilitate certain actions.

Based on our survey results [4], we can conclude there is a growing interest from users to have more control over their location data. However, challenges in achieving widespread adoption and implementation of our proposed system by industry players remain. Interoperability relies on the use of specifications, which we can only encourage, but not enforce. Data sovereignty can help steer this in the right direction, but ultimately, it is up to the industry players to decide to adopt and implement these specifications. Future work should investigate how to facilitate the structuring of this data without putting this responsibility on the client that implements our integrated solution.

From our survey results, we also gathered that users are concerned about their location data being stolen in a data breach (Appendix E.2). In this integrated solution, we propose an architecture that moves data to personal data vaults managed by the user. Users can provide access to their data through our implementation that uses the Solid project as the personal data vault. However, security concerns remain valid. Related work investigated the security and privacy aspects of Solid [254, 255]. In this related work, general issues were identified, such as the responsibility of the Pod provider to adhere to security standards. One prominent issue that remains in Solid is the lack of data minimisation, which prevents the sharing of anonymised data. In the context of positioning systems, Location Privacy Protection Mechanisms (LPPMs) should apply to the data before sharing it with other parties.

By developing modular frameworks such as OpenHPS that provide tools for creating positioning systems and optionally add modules for ensuring their interoperability, we steer the industry in the right direction without forcing them to. Similarly, our SemBeacon solution is not limited only to indoor positioning systems. Use cases such as the discovery of IoT devices help to expand the potential scope of adoption for our proposed system.

Chapter 8

Discussion and Future Work

Our initial goal when starting this research was to design a method for positioning systems to seamlessly work together to track spatial objects or individuals. With a shift in the perspective on the privacy of location data in the last decade, we also wanted to ensure that individuals could control and see which systems were tracking them.

The design of such systems requires a more thorough understanding of the various technologies that accommodate location tracking, as well as the methods to make each technology and system work together. This led to the design of interoperable positioning systems, systems that can seamlessly work together with other interoperable systems to track the same object or individual, while potentially relying on different technologies. In this dissertation, we explained the concept of such systems and why it is necessary to enable the discovery of these systems.

We designed a solution that provides users and objects with their own data space that contains personal data, including sensor output and computed information about their position. These data vaults are private but can be accessed when users allow third-party applications to access and modify the data. Data is stored as linked data, enabling the reasoning of this data. Positioning systems, whether it is an indoor positioning system or a simple system that tracks objects on a meeting table also have a personal or public data space that contains information about the capabilities of the system, as well as a semantic description of the inner workings of this system and how input data is processed to a certain output.

By leveraging the Semantic Web, we provided a foundation that also enables these indoor positioning systems to be used for inference-based positioning or decision-based sensor fusion. This type of fusion, detailed in Section 2.1.1, offers the ability to infer a location based on additional user-context well beyond what is currently envisioned in our architecture. Our solution can integrate with future context brokers such as Next Generation Service Interfaces – Linked Data (NGSI-LD) [256]

to enrich other services and obtain additional contextual information for tracking subjects.

For the discovery of positioning systems, we rely on their physical discovery by letting the infrastructure of the positioning systems announce their availability to users (e.g., via the Wi-Fi access points, Bluetooth beacons or even a QR code for registering the system). Users can choose to provide access to their data space for the positioning system or even revoke this access when they no longer want to be tracked by a certain application or system. By storing all data about users in a user-centric location, positioning systems can work together to track the same object and enable seamless handover from one tracking method to another.

8.1 Discussion

We summarised how our integrated solution provides an answer to our three main research questions to create interoperable and discoverable positioning systems. In the following, we reflect on the limitations of this integrated solution and other research artefacts that contributed to this solution.

Looking at the individual aspects and research questions of this dissertation, we first started with RQ1 that aimed to generalise hybrid positioning systems. We subdivided this research question into the generalisation of input data (RQ1.1) and output data (RQ1.2) as we need to define what input and output a *hybrid positioning system* consumes and produces to generalise it. In our research, we provided a solution to this research question by developing our OpenHPS framework, and we introduced a modular data structure that can be applied to a wide range of use cases.

OpenHPS was developed in TypeScript, which limits its computational performance compared to other — more low-level programming languages. Our framework was designed with data generality in mind to pave a path towards interoperability. Positioning systems may often use algorithms that heavily rely on computer vision or, in more recent examples, machine learning. TypeScript was chosen as it enables OpenHPS to be deployed in web browsers, servers, mobile applications, as well as some embedded devices. We demonstrated using a benchmark and available modules that we can optimise OpenHPS to work on multiple workers and systems. We also discussed that we can create bindings with low-level frameworks such as OpenCV to handle computational tasks, while still using the data structure in OpenHPS to handle the processing of the high-level data.

While our decision to represent positioning systems as graph processing networks offers a flexible framework, some procedures or algorithms are difficult to implement in a process network. Actions that are performed by a user, such as authentication or the calibration of a sensor, are blocking actions that are difficult to implement in a pipeline. To solve this issue, we introduced a wide range of

services that help to perform these tasks in the background. While this solves our issue, it does introduce additional complexity in the design of a positioning system.

From the feedback we received from industry players, OpenHPS offers a backbone for designing a wide range of positioning systems. However, the graph-based architecture and low-level aspect of choosing your own algorithms make it difficult to implement a system without prior experience. As outlined in the introduction of Chapter 3, our framework is aimed at developers and researchers with knowledge of the algorithms used within a positioning system. Future work should expand on OpenHPS to provide additional layers of abstractions that offer pre-built graphs or GUIs to enable end users to design the targeted systems. These graphs would still be modifiable, but by adding different layers of abstraction, we can adapt our interface to the end user's expertise. Furthermore, to facilitate the adoption of OpenHPS and the modules that provide interoperability, more efforts should be made to offer pre-built applications and tools. Currently, developers wishing to use OpenHPS have to develop everything from scratch, even if it is a simple application that helps in collecting fingerprints or other sensor data. Providing these applications to developers will already help in the adoption of the framework.

Next, after defining a representation of a hybrid positioning system and generalising its data, we investigated the interoperability of these systems to answer RQ2. Interoperability was defined as the ability for multiple applications to *access*, *read* and *understand* the data belonging to a user. We proposed a solution to RQ2.1 by using personal data vaults that tackle the access and reading of the data. We use linked data to describe positioning systems and the data they produce. To answer RQ2.2, we developed the POSO ontology to semantically describe these systems, enabling us to perform reasoning on the data they produce.

In Section 2.4.3, we addressed the issue of the additional overhead involved with working with linked data. While it offers a rich way to describe and reason about positioning systems, the use of linked data also introduces complexity in terms of data processing and storage. One limitation of our approach is the potential scalability issues that may arise when dealing with large amounts of linked data, especially in real-time positioning scenarios where data is constantly being generated and processed. This issue is not only related to positioning systems but is a hot topic amongst researchers in the field of the Semantic Web. We already worked towards the scalability issues by leveraging specifications such as Linked Data Event Stream (LDES), which allows for more efficient data streaming and incremental updates to linked data resources. However, further research and development are needed to fully address the scalability challenges posed by linked data in the context of positioning systems.

We acknowledge that our integrated solution, using Solid as its current implementation to provide user-centric storage of data, is not efficient enough for real-time storage, real-time collaboration and real-time sensor fusion. However, the concep-

tual design of our architecture, along with the syntactic, semantic and processing interoperability that we offer to positioning systems, provides a stable foundation to improve upon. With Solid still being actively developed and improved, we foresee that it will offer future researchers the possibility to enable more efficient storage and retrieval of data. In turn, this will ensure that the user-centric storage becomes a communication broker for enabling high-level sensor fusion and collaboration of positioning systems.

While interoperability is a good start to enable multiple applications to access positioning systems or their data, they should be able to discover the existence of these systems. To provide a solution for RQ3 and RQ3.2, we created SemBeacon, a semantic beacon specification for advertising resources via Bluetooth Low Energy. Our solution enables users to discover semantic resources in physical environments based on their proximity to these semantic beacons.

While the use of a custom specification enables the discovery, it could break the interoperability aspect of the solution, as it requires a proprietary technology and detection algorithm. With the design of our SemBeacon solution, we have built on existing specifications such as AltBeacon and Eddystone-URL. In our technical evaluations of SemBeacon, we provide various prototypes built on top of existing libraries that can already scan for beacons that implement these existing specifications. With research question RQ3.1, we specifically focus on the interoperability of the discovery. We enabled interoperability by providing the ability for SemBeacon to advertise standardised linked data resources, making it easier for different applications to understand and interact with the discovered resources.

SemBeacons offer local discovery of positioning systems by physically transmitting a URI when near such a system. However, due to the requirement of specific hardware and the interoperability issues that may result from the use of specific communication protocols, we also designed *Linked Data Hash Tables*. These hash tables offer a distributed approach to data sharing via an interoperable semantic description of this data. Similar to our decentralisation with Solid, LDHT uses the Solid specification to offer the creation of distributed hash tables. While this specification is available in OpenHPS, it is currently still under development to make it more scalable and requires more intensive benchmarking to test its performance over a larger set of nodes.

The integrated solution we have presented in Chapter 7 combines the answers to all the research questions in a solution that strongly relies on personal data vaults as the storage of input and output data of positioning systems. In our proposed design, we have used the Solid project [139] to act as the personal data vault due to its use of linked data, which we already defined for RQ2 as a semantic method to enable interoperability. To discover these systems and the subjects that are tracked by these systems, we use SemBeacons to advertise Web Identifiers (see Section 7.6) to uniquely identify subjects in physical environments.

One of the key challenges that remains is finding the balance between interoperability, performance, scalability, and security in the integrated solution. Solutions such as fragmenting the data with Linked Data Event Stream (LDES) as proposed in our integrated solution in Section 7 ensure that data can be relatively efficiently distributed and processed in real time, but it also introduces additional complexity for applications and positioning systems to adhere to this structure. Furthermore, this additional complexity becomes problematic for the user or tracked subject when they want to manage the access over which consumers can access part of their location data. One possible solution to this problem would be to add a layer of abstraction over the personal data vaults that adheres to certain specifications and standardisations to store data. Such an abstraction layer could remove the responsibility of storing interoperable data while ensuring that the performance and consistency of the stored data can be maintained. Van Herwegen and Verborgh [240] proposed a method for fine-grained access control through endpoints that abstract the execution of SPARQL queries. Such a system could be applied to location data with multiple endpoints abstracting the underlying data structure.

Another open question that remains is for systems to *request* access to a user's location data. In our current integrated solution, user-initiated tracking allows users to *provide* access to a positioning system, for it to store data in their personal data vault. However, in system-initiated tracking, a user may not be aware that a positioning system wants to track them. To solve this, a positioning system should request access to their data. One possible solution to this problem would be to create a public *inbox* where unauthorised positioning systems (i.e., agents) can notify the user that they request access.

8.2 Future Work

In this dissertation, we paved the way towards the interoperability of indoor positioning systems. Hybrid- or integrated positioning systems can contain a wide range of positioning techniques. The graph-based processing network we developed with OpenHPS in Chapter 3 was validated with major positioning system architectures and technologies. However, with the increase of research on new methods of positioning, which often rely on large trained datasets, this stream-based processing should be validated with modern algorithms. We have already started the validation with the release of modules for enabling artificial intelligence within processing networks[1], but this module and the data it represents are currently not mapped to linked data. Ontologies such as the Neural Network ontology [132] could be used to represent procedures that rely on pre-trained neural networks.

The generic Positioning System Ontology (POSO) introduced in Section 4.2 offers a stable core ontology for describing positioning systems and the data they produce.

[1] `https://openhps.org/docs/tf/`

While it is designed to be as generic as possible so it can be applied to a wide range of positioning systems, current examples and applications are heavily focused on indoor positioning systems. Future work should investigate if the POSO ontology is generic enough to apply to more advanced positioning systems and is also descriptive enough to ensure the interoperability of more advanced algorithms, such as computer vision.

Our solution heavily relies on the Solid project, which is still in its early stages of development. As such, future work might involve further collaboration with the Solid community to enhance the functionalities and scalability of our solution. In its current state, the main downside of Solid and the semantic linked data it would contain is the overhead required to query information within a Pod, without knowledge of the shape or structure of the linked data [219]. Furthermore, the Solid specification draft aims to offer a solution for persisting and synchronising offline changes. Such offline changes would enable positioning systems to store real-time data locally without requiring a constant network connection. However, no implementations have currently been made in Solid to enable this functionality and it is therefore not considered in our integrated solution.

The work that was presented in this thesis provides a foundation for future research and industrial applications in the field of indoor positioning systems. With users becoming more and more aware of their privacy and new regulations being introduced to address these concerns, our solution offers a baseline to build on. In our solution, we have addressed Objective 2 by investigating the seamless data exchange and reusability of indoor positioning systems. What is left is an effort from those systems to accept that seamless data exchange is necessary and beneficial. Companies such as Google have already started migrating away from cloud-based location-based services since 2024 to adhere to regulations[2], but their solution currently relies on storing raw location data locally on a user's device. This limits the availability and accessibility of the data to other services or even other devices. Our solution would provide both companies and users with the seamless storage and exchange of location data while also ensuring the expected privacy, transparency and control that goes along with this data.

One possible solution to unravel the issue of data producers having too much responsibility to adhere to the interoperable architecture is to introduce an orchestrator in a personal data vault. This orchestrator is managed by the user and ensures that data producers adhere to the specifications of the linked data architecture by restructuring data and applying policies such as retention and storage policies. In addition, this orchestrator can create an abstraction around the data vault that *provides* the data with different granularities without having to replicate the actual data. This, in turn, can pave a path towards a more optimised storage of data.

[2] https://support.google.com/maps/answer/14169818?hl=en

Finally, with the contributions in this dissertation, we focused on the technical aspect of ensuring that we can create interoperable and discoverable indoor positioning systems. To scope our design, we did not investigate how we can ensure the correctness of the data published in the Pod of a user. Future work should determine how users can verify that the data produced by positioning systems is correct. Furthermore, a consumer of this data should also be given assurances on the correctness of the data. For example, a navigation application that makes use of the location data to determine how busy an area is should be aware if a location is altered or fake.

With the open specifications, frameworks and examples that we created during our research, including OpenHPS[3], SemBeacon[4], POSO[5] and LDHT, we hope that future researchers and developers can continue the creation of interoperable positioning systems and facilitate their discovery by non-proprietary applications. This way, we hope that one day we have *Findable, Accessible, Interoperable and Reusable (FAIR)* indoor positioning systems to guide us to our destination.

[3] https://openhps.org
[4] https://sembeacon.org
[5] https://poso.openhps.org

Appendix A

OpenHPS

This appendix contains technical information about the OpenHPS framework and its modules. An up-to-date version can be found on the online documentation[1].

A.1 UML

A.1.1 Data Objects

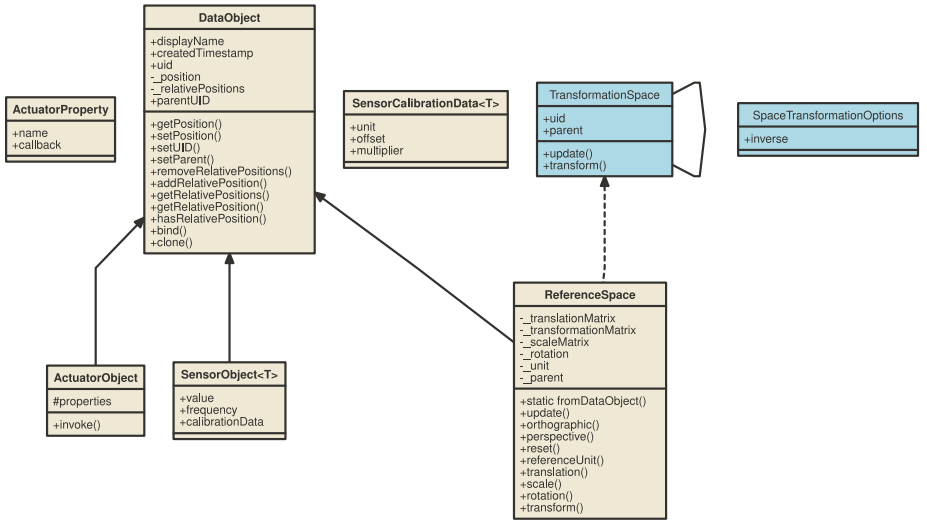


Figure A.1: UML diagram of the main data objects in the core of OpenHPS

[1] https://openhps.org/docs/core/

A.1.2 Sensor Objects

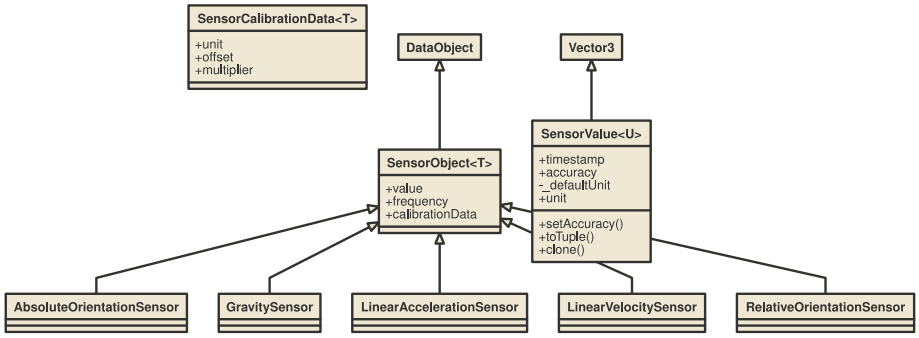


Figure A.2: UML diagram of the main sensor objects in the core of OpenHPS

A.1.3 Sensor Values

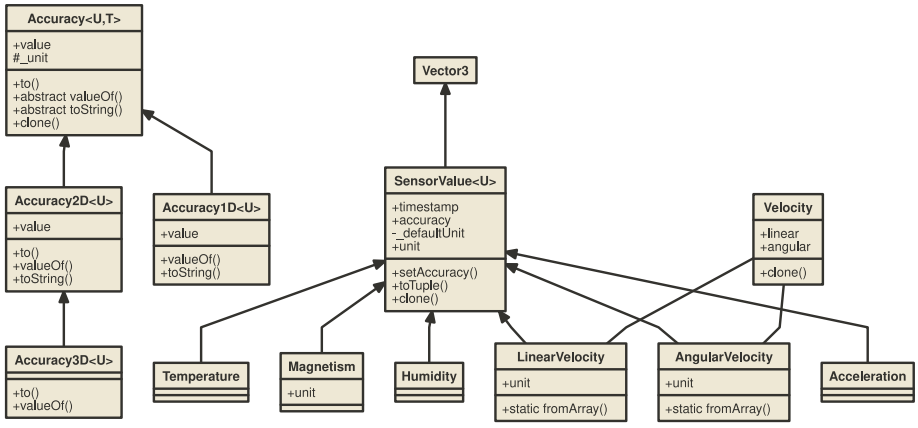


Figure A.3: UML diagram of the main sensor values in the core of OpenHPS

A.1.4 Absolute and Relative Position

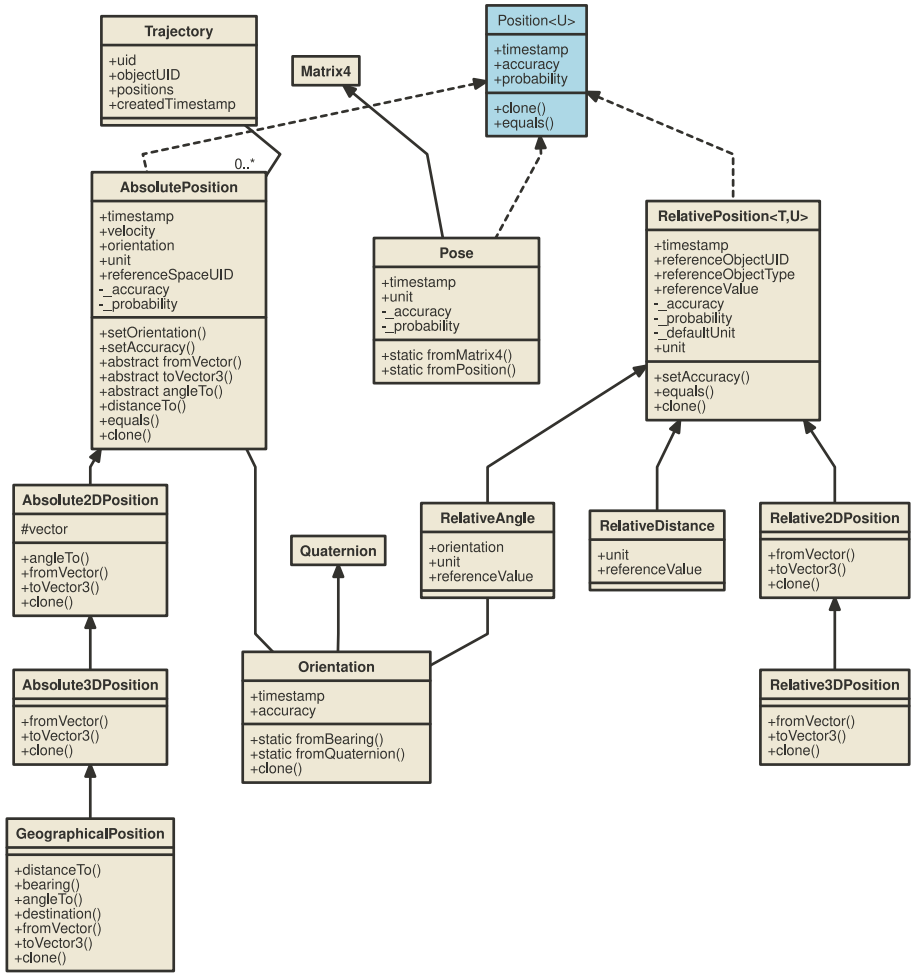


Figure A.4: UML diagram of the absolute and relative position classes

A.1.5 Graph

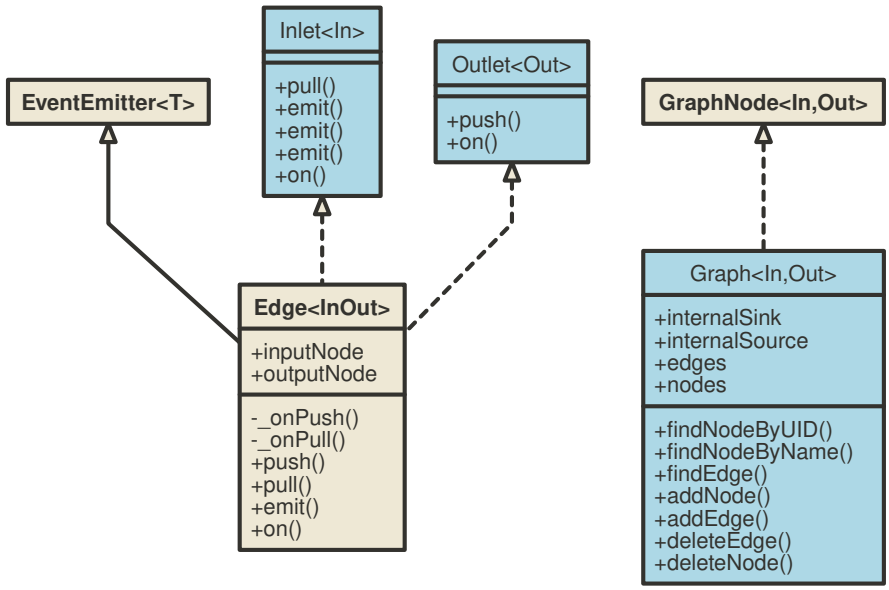


Figure A.5: UML diagram of the graph-related classes

A.1.6 Services

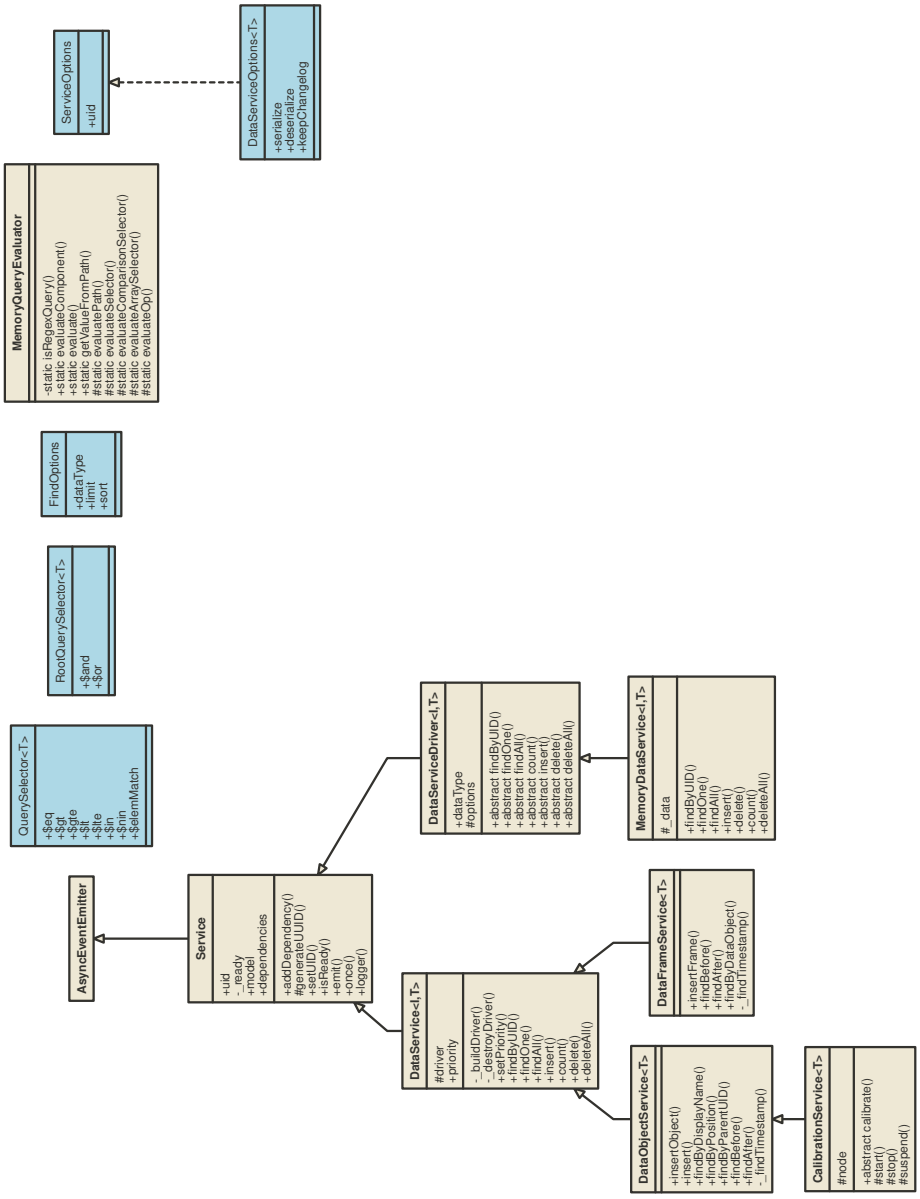


Figure A.6: UML diagram of the service-related classes

A.1.7 Units

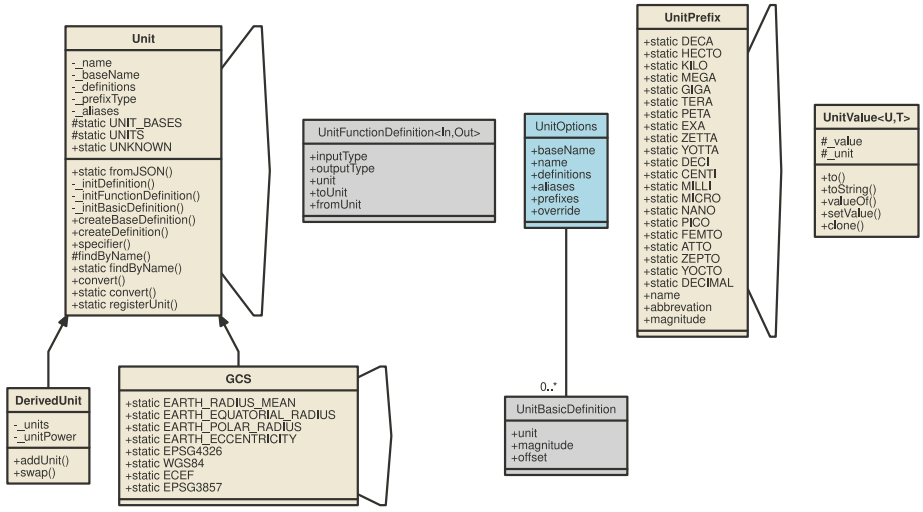


Figure A.7: UML diagram of the unit-related classes

A.2 Dependencies

In Chapter 3, we detailed the use of TypedJSON as one of our main dependencies for serialising data. However, one of the other main dependencies in the framework is the Three.js framework. Three.js provides mathematical concepts for representing matrices, vertices and quaternions, which we utilise within OpenHPS. All dependencies in OpenHPS either have an MIT or Apache-2.0 license, which ensures that the framework remains open-source and free for anyone to use and modify. Additionally, Three.js is widely used and well-documented, making it easier for developers to understand and contribute to the OpenHPS framework.

Figure A.8 shows the tree map of all components in the core module of OpenHPS. The Web module has a total size of 309KB. In total 43.4% of the final bundle amounts to dependencies. The extension on top of TypedJSON along with the data structure amounts to 18.9% and the built-in nodes amount to 12.1%.

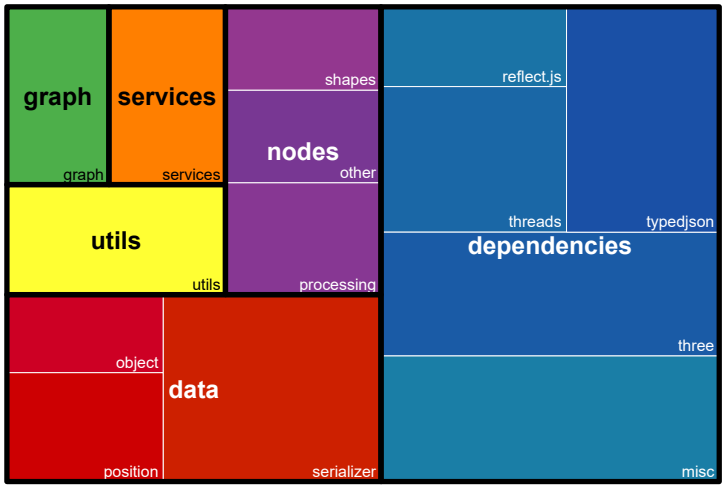


Figure A.8: Tree map of the components in our minified OpenHPS Core module

A.3 Examples

A.3.1 Fiducal Markers as Reference Spaces

```
@SerializableObject({
  rdf: { type: fidmark.FiducialMarker }
})
export class FiducialMarker extends ReferenceSpace {
  @SerializableMember({
    rdf: {
      predicate: fidmark.markerData, datatype: xsd.string
    },
  })
  data?: string;
  @SerializableMember({
    rdf: {
      predicate: fidmark.markerIdentifier, datatype: xsd.integer
    }, numberType: NumberType.INTEGER
  })
  identifier?: number;
  @SerializableMember({
    rdf: { predicate: fidmark.hasDictionary }
  })
  dictionary?: MarkerDictionary;
  origin?: MarkerOrigin;

  @SerializableMember({
    rdf: {
```

```
25        predicate: [fidmark.hasHeight],
26        serializer: (value: number) => {
27          return RDFBuilder.blankNode()
28            .add(rdf.type, qudt.QuantityValue)
29            .add(qudt.unit, LengthUnit.MILLIMETER)
30            .add(qudt.numericValue, value, xsd.double).build();
31        },
32        deserializer: (thing: Thing) => {
33          const unit = RDFSerializer.deserialize(
34              thing.predicates[qudt.unit][0] as Thing, LengthUnit);
35          return unit.convert(parseFloat(
36              thing.predicates[qudt.numericValue][0].value),
                ↪  LengthUnit.MILLIMETER);
37        },
38      },
39    })
40    height?: number;
41    @SerializableMember({
42      rdf: {
43        predicate: [fidmark.hasWidth],
44        serializer: (value: number) => {
45          return RDFBuilder.blankNode()
46            .add(rdf.type, qudt.QuantityValue)
47            .add(qudt.unit, LengthUnit.MILLIMETER)
48            .add(qudt.numericValue, value, xsd.double).build();
49        },
50        deserializer: (thing: Thing) => {
51          const unit = RDFSerializer.deserialize(
52              thing.predicates[qudt.unit][0] as Thing, LengthUnit);
53          return unit.convert(parseFloat(
54              thing.predicates[qudt.numericValue][0].value),
                ↪  LengthUnit.MILLIMETER);
55        },
56      },
57    })
58    width?: number;
59    @SerializableMember({
60      rdf: { predicate: fidmark.hasImageDesciptor }
61    })
62    imageDescriptor?: ImageDescriptor;
63  }
```

Listing A.1: Fiducial marker object created as a `ReferenceSpace` in OpenHPS

A.3.2 Beacon Classification

During our development of SemBeacon, we have expanded the capabilities of our RF module (`@openhps/rf`) to facilitate the development of use cases where unknown beacons had to be *discovered*. One of our additions is a processing node that can classify the beacon type based on raw advertisement data. After providing a list of the different beacon types that can be detected, the node will match the

```
import { ModelBuilder, DataFrame, CallbackSinkNode } from '@openhps/core';
import {
  BLEBeaconClassifierNode, RelativeRSSIProcessing, PropagationModel,
  BLEAltBeacon, BLEiBeacon,
  BLEEddystoneURL, BLEEddystoneUID, BLEEddystoneTLM,
} from '@openhps/rf';
import { BLESourceNode } from '@openhps/capacitor-bluetooth';

ModelBuilder.create()
  .from(new BLESourceNode({ uid: "ble" }))
  .via(new BLEBeaconClassifierNode({
    resetUID: true,
    types: [ BLEAltBeacon, BLEiBeacon, BLEEddystoneURL,
      BLEEddystoneUID, BLEEddystoneTLM ]
  })).via(new RelativeRSSIProcessing({
    environmentFactor: 2.0,
    propagationModel: PropagationModel.LOG_DISTANCE
  })).to(new CallbackSinkNode((frame: DataFrame) => {
    // ...
  }));
```

Listing A.2: Scanning and classifying beacons in OpenHPS

beacon that matches first. Developers can easily create new beacon objects in OpenHPS as data objects that extend a `BLEBeaconObject`.

A.3.3 Protocol Buffers

The `@openhps/protobuf` module uses *protocol buffers* [184] to limit the amount of bandwidth used to communicate data via MQTT, socket connections or other communication layers. Protocol buffers work by defining a *protocol* beforehand that indicates which fields will be included in a message, what data type these fields contain, and also which other messages will be sent in the same protocol.

OpenHPS provides a data structure for data objects, data frames, and other data that is relevant to a positioning system. However, the data itself is not fixed which allows developers or other modules to extend these data types. While this offers flexibility, it also prevents the use of a single protocol for the entire OpenHPS framework which is the main reason why JSON or RDF data is used to serialise data.

In `@openhps/protobuf`, we solve this issue by automatically generating protocol messages based on the *decorated* (see Section 4.3) objects and field names. At runtime, we can access the decorators along with the data types to automatically generate a protocol as shown in Listing A.3 for a data object and absolute position.

Listing A.4 demonstrates how developers can then integrate these protocol buffers in communication layers such as MQTT. Each communication module allows

```
package openhps.core;
syntax = "proto3";
import "google/protobuf/any.proto";
message DataObject {
        string displayName = 1;
        int64 createdTimestamp = 2;
        string uid = 3;
        string parentUID = 4;
        google.protobuf.Any position = 5;
        repeated google.protobuf.Any relativePositions = 6;
}

package openhps.core;
syntax = "proto3";
import "Velocity.proto";
import "Orientation.proto";
import "google/protobuf/any.proto";
message Absolute3DPosition {
        string timestamp = 1;
        Velocity velocity = 2;
        Orientation orientation = 3;
        google.protobuf.Any unit = 4;
        string referenceSpaceUID = 5;
        google.protobuf.Any accuracy = 6;
        string probability = 7;
        double x = 8;
        double y = 9;
        double z = 10;
}
```

Listing A.3: Generated protobuf message for a `DataObject` and `Absolute3DPosition`

```
ModelBuilder.create()
  .addService(new MQTTServer({
    port: 1443,
  }))
  .from(new MQTTSourceNode({
    uid: "source",
    // Override frame serializer (not the options)
    serialize: (obj, options) => ({
      frame: ProtobufSerializer.serialize(obj),
      options
    }),
    deserialize: (obj) => ProtobufSerializer.deserialize(obj.frame)
  }))
  .to()
  .build();
```

Listing A.4: Integrating protocol buffers in MQTT

the override of the serialisation and deserialisation functions, which are utilised to replace the default JSON serialiser with the more optimised protocol buffers. While this ensures a more efficient data exchange, it removes all contextual information from the data, which requires each node to have the same version of the generated protocol.

A.3.4 Data Owner in Solid

Linking a user to a positioning system is the first step towards user-centric data storage for positioning systems. However, internally, a positioning system needs to manage multiple users.

In Chapter 3 we defined the concepts of `DataObject` and `DataFrame`. We detailed that we use these concepts to cover all data produced and consumed by a positioning system. Our `@openhps/solid` module *augments* a new property for all data objects to include a Web Identifier (WebID). Listing A.5 illustrates how a WebID can be set for a data object to link objects to one user.

```
import { DataObject } from '@openhps/core';

const phone = new DataObject("myphone");
phone.webId = "https://solid.maximvdw.be/profile/card#me";
```

Listing A.5: Indicating an owner of a data object in OpenHPS

A.4 Garage Fingerprinting Dataset

In 2020, during the development of OpenHPS, we scheduled the creation of a large crowdsourced dataset as part of our ongoing research towards the crowdsourcing of fingerprinting data. Due to the COVID-19 pandemic, this was not feasible until 2021 when we created a dataset containing Wi-Fi, Bluetooth and IMU data [35].

However, to continue development, a smaller-scale dataset was created in a garage. The raw unprocessed dataset was used for 5 years during the course *Information Visualisation* to teach students how to process and extract the information, and was a stepping stone towards our final dataset and mobile application to capture this dataset. While the dataset was only released publicly in 2025, it served as a valuable resource for testing and refining the OpenHPS framework with its use in integration testing of the `@openhps/imu` and `@openhps/rf` modules.

Figure A.9: Dataset recording environment

Figure A.9 depicts the environment in which the recording was made. Data was captured in a grid with each point spanning 50 cm apart. Four Bluetooth beacons were placed around the garage and were part of the recorded data along with IMU and WLAN data. Figure A.10 visualises the RSSI readings of a beacon placed in a corner. While the signal propagation can be distinguished from the recorded readings, the dataset is too small to perform meaningful positioning. Our full dataset is available on Kaggle [38].

A.4.1 Impact

The recording of this dataset helped us optimise the process of recording such data. The following issues were addressed as a response to this dataset:

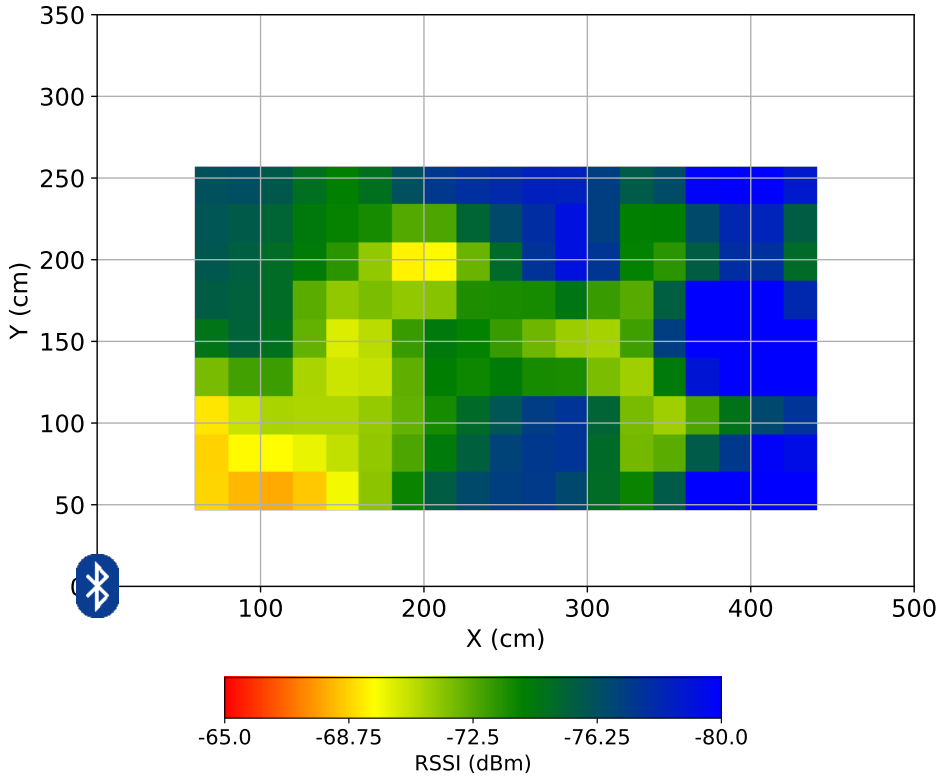


Figure A.10: Beacon RSSI readings

- **Grid size**: Despite only using 45 data points, the recording of the dataset took a very long time. With this dataset, we wanted to record fingerprint data in four directions. Due to the update frequency of the Wi-Fi scanning, we also wanted to have a 30 second recording for each orientation. Combined with the setup and all other tasks related to the creation of the dataset, the recording took well over a day for a single room. In a dataset that spans an entire building or floor, this would not be feasible.
- **User input**: Our initial application to record the sensor data required user input for the coordinates. This increased the likelihood of human error in data collection and also introduced more complexity in the recording process. The application was later modified to visualise the unrecorded data points.
- **Orientation**: Our recording application had no user input for the orientation in which the recording was made. Instead, the orientation was based on the compass direction of the smartphone. While this was sufficient for our dataset, it resulted in more post-processing to normalise these orientations. Additionally, it also introduced human error as there was no feedback to the

user on which orientation was completed. This human error also led to the dataset being recorded twice. In our application, we made a clear UI that shows which orientation was missing, similar to our changes that visualise unrecorded data points. Another issue related to the orientation was the bias introduced by the motor of the garage door. While post-processing the data, we noticed that our compass drifted towards the motor in certain areas.

- **Constant data collection**: With our first version of the recording application, we collected data non-stop from the beginning of the session until the end. Only during post-processing, we segmented the data and removed data in between data points. This segmentation was accomplished by utilising the accelerometer data to determine if the user was standing still after inputting the position. However, this post-processing was too complex. Moreover, the constant collection of data, especially IMU data resulted in more than 20 GB of unprocessed data. The mobile application would also crash when network conditions were not ideal, which was a likely problem in our large-scale dataset. To solve this issue, we relied again on the visualisation of unrecorded data points. By clicking on such a data point, the application would only collect and transmit sensor data for a specific amount of time.
- **Usefulness of trajectory data**: While our constant data collection was not ideal for the stability of our recording, it did highlight a feature that would be nice for our final large-scale dataset. Trajectory data in combination with fingerprint data would not only provide researchers with valuable spatial information but also information on the movement patterns within the environment. This led us to include trajectory data in our final dataset recording application.
- **Usefulness of Wi-Fi access points**: During the pseudonymisation of the dataset, more specifically, the MAC addresses of the Wi-Fi access points, we noticed that a lot of contextual data was missing. To strengthen our final dataset, we included additional pseudonymised context about each access point.
- **Training and testing**: The limited number of data points made it difficult to perform effective training and testing of algorithms for indoor positioning. In our next dataset, we added specific test data points in between training data points.
- **Improving CSV handling**: Our dataset was internally kept as JSON files with the serialised data from OpenHPS. After processing, we needed an effective method to handle the data. To aid in the handling of such data, we modified `@openhps/csv` to help developers use datasets for developing positioning systems.

- **Magnetometer calibration**: Magnetic positioning uses the magnetometer sensor inside a smartphone. These signals are susceptible to *hard iron* and *soft iron* effects. To address this issue in our next dataset, we normalised the magnetometer data after calibrating it using existing calibration techniques. This ensured that the magnetic positioning data in our dataset was more accurate and reliable for future research and development.

Overall, the dataset helped us to prepare for the large-scale dataset that we recorded one year later. While this dataset was not as detailed as our final dataset, it still proved itself useful and is still used in integration tests throughout the OpenHPS framework.

Appendix B

POSO

B.1 Version 1.0

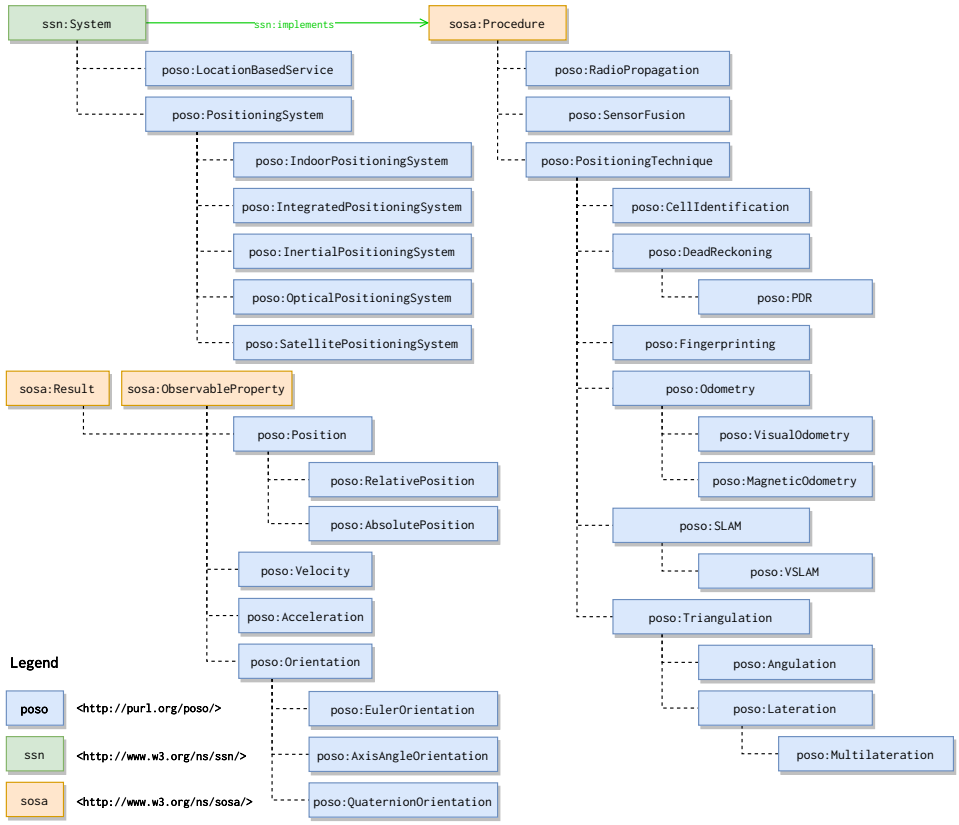


Figure B.1: Main POSO ontology classes in version 1.0

Figure B.1 illustrates the structure of the main classes in version 1.0 of the POSO ontology. An updated illustration can be found in Figure 4.4 in Section 4.2. The dotted line in the figure represents the `rdfs:subClassOf` relationship.

B.2 Extensions

B.2.1 Common Positioning Systems and Algorithms

```
#################################################################
#    Object Properties
#################################################################

###  http://purl.org/poso/common/hasReferenceRSSI
:hasReferenceRSSI rdf:type owl:ObjectProperty ;
  rdfs:comment "has a reference Received Signal Strength Indicator at a
  ↪  specified distance."@en ;
  rdfs:label "has reference RSSI"@en .

#################################################################
#    Data properties
#################################################################

###  http://purl.org/poso/common/major
:major rdf:type owl:DatatypeProperty ;
  rdfs:domain :iBeacon , [ rdf:type owl:Restriction ;
   owl:onProperty :major ;
   owl:someValuesFrom xsd:integer ] ;
  rdfs:range xsd:int ;
  rdfs:label "major"@en .

###  http://purl.org/poso/common/minor
:minor rdf:type owl:DatatypeProperty ;
  rdfs:domain :iBeacon , [ rdf:type owl:Restriction ;
   owl:onProperty :minor ;
   owl:someValuesFrom xsd:integer ] ;
  rdfs:range xsd:int ;
  rdfs:label "minor"@en .

###  http://purl.org/poso/common/proximityUUID
:proximityUUID rdf:type owl:DatatypeProperty ;
  rdfs:domain [ rdf:type owl:Restriction ;
      owl:onProperty :proximityUUID ;
      owl:someValuesFrom xsd:hexBinary
    ] ;
  rdfs:range [ rdf:type rdfs:Datatype ;
     owl:onDatatype xsd:hexBinary ;
     owl:withRestrictions ( [ xsd:maxLength "32"^^xsd:int ] )
       ] ;
  rdfs:label "proximity UUID"@en .

#################################################################
#    Classes
#################################################################

```

```
### http://purl.org/poso/common/AltBeacon
:AltBeacon rdf:type owl:Class ;
  rdfs:subClassOf poso:BluetoothBeacon ;
  rdfs:comment "AltBeacon is an open and interoperable proximity beacon
↪ specification."@en ;
  rdfs:label "AltBeacon"@en ;
  rdfs:seeAlso "https://altbeacon.org/"^^xsd:anyURI .

### http://purl.org/poso/common/EddystoneBeacon
:EddystoneBeacon rdf:type owl:Class ;
  rdfs:subClassOf poso:BluetoothBeacon ;
  rdfs:comment "Eddystone was a Bluetooth Low Energy beacon profile
↪ released by Google in July 2015."@en ;
  rdfs:label "Eddystone"@en ;
  rdfs:seeAlso <http://dbpedia.org/resource/Eddystone_(Google)> .

### http://purl.org/poso/common/EddystoneEID
:EddystoneEID rdf:type owl:Class ;
  rdfs:subClassOf :EddystoneBeacon ;
  rdfs:label "Eddystone-EID"@en .

### http://purl.org/poso/common/EddystoneTLM
:EddystoneTLM rdf:type owl:Class ;
  rdfs:subClassOf :EddystoneBeacon ;
  rdfs:comment "Eddystone-TLM provides telemetry information. It can
↪ include the battery voltage, temperature, advertisement count and
↪ uptime."@en ;
  rdfs:label "Eddystone-TLM"@en .

### http://purl.org/poso/common/EddystoneUID
:EddystoneUID rdf:type owl:Class ;
  rdfs:subClassOf :EddystoneBeacon ;
  rdfs:comment "Eddystone UID provides a 10 byte namespace and 6 byte
↪ instance identifier."@en ;
  rdfs:label "Eddystone-UID"@en .

### http://purl.org/poso/common/EddystoneURL
:EddystoneURL rdf:type owl:Class ;
  rdfs:subClassOf :EddystoneBeacon ;
  rdfs:comment "Eddystone-URL can broadcast a short encoded URL."@en ;
  rdfs:label "Eddystone-URL"@en .

### http://purl.org/poso/common/SemBeacon
:SemBeacon rdf:type owl:Class ;
  rdfs:subClassOf poso:BluetoothBeacon ;
  rdfs:comment "A SemBeacon is a semantic beacon based on AltBeacon and
↪ Eddystone-URL."@en ;
  rdfs:label "SemBeacon"@en ;
  rdfs:seeAlso <http://purl.org/sembeacon/SemBeacon> .

### http://purl.org/poso/common/iBeacon
:iBeacon rdf:type owl:Class ;
  rdfs:subClassOf poso:BluetoothBeacon ;
```

```
93    rdfs:comment "An iBeacon is a beacon that advertises the proximity beacon
      ↪  packet specification."@en ;
94    rdfs:label "iBeacon"@en ;
95    rdfs:seeAlso "https://developer.apple.com/ibeacon/"^^xsd:anyURI .
96
97  ###################################################################
98  #    Individuals
99  ###################################################################
100
101 ###  http://purl.org/poso/common/AbsolutePositionOutput
102 :AbsolutePositionOutput rdf:type owl:NamedIndividual , poso:PositionOutput
      ↪  , rdfp:GraphDescription ;
103   rdfs:label "Absolute position output"@en ;
104   rdfp:presentedBy [ ] .
105
106 ###  http://purl.org/poso/common/Anyplace
107 :Anyplace rdf:type owl:NamedIndividual , poso:PositioningPlatform ;
108   rdfs:comment "Anyplace is a first-of-a-kind indoor information service
      ↪  offering GPS-less localization, navigation and search inside buildings
      ↪  using ordinary smartphones."@en ;
109   rdfs:isDefinedBy <https://anyplace.cs.ucy.ac.cy/> ;
110   rdfs:label "Anyplace"@en .
111
112 ###  http://purl.org/poso/common/BDS
113 :BDS rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
114   rdfs:comment "BeiDou, or BDS, is a global GNSS owned and operated by the
      ↪  People's Republic of China. BDS was formally commissioned in 2020. The
      ↪  operational system consists of 35 satellites. BDS was previously called
      ↪  Compass."@en ;
115   rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
116   rdfs:label "BeiDou Navigation Satellite System"@en ;
117   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
118
119 ###  http://purl.org/poso/common/CSIFingerprinting
120 :CSIFingerprinting rdf:type owl:NamedIndividual , poso:Fingerprinting ;
121   rdfs:label "Channel State Information fingerprinting"@en .
122
123 ###  http://purl.org/poso/common/EarthDeployment
124 :EarthDeployment rdf:type owl:NamedIndividual , poso:OutdoorDeployment ;
125   rdfs:comment "A generic deployment for any positioning system deployment
      ↪  on Earth."@en ;
126   rdfs:isDefinedBy <https://dbpedia.org/resouce/Earth> ;
127   rdfs:label "Earth deployment"@en .
128
129 ###  http://purl.org/poso/common/FootMountedPDR
130 :FootMountedPDR rdf:type owl:NamedIndividual , poso:PDR ;
131   rdfs:comment "Foot-mounted pedestrian dead reckoning uses a sensor on the
      ↪  foot of a pedestrian to detect steps."@en ;
132   rdfs:label "Foot-mounted pedestrian dead reckoning"@en .
133
134 ###  http://purl.org/poso/common/GLONASS
135 :GLONASS rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
```

```
136   rdfs:comment "GLONASS (Globalnaya Navigazionnaya Sputnikovaya Sistema, or
      ↪  Global Navigation Satellite System) is a global GNSS owned and operated
      ↪  by the Russian Federation. The fully operational system consists of 24+
      ↪  satellites."@en ;
137   rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
138   rdfs:label "Global Navigation Satellite System"@en , "Globalnaya
      ↪  Navigazionnaya Sputnikovaya Sistema"@ru ;
139   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
140
141 ###  http://purl.org/poso/common/GPS
142 :GPS rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
143   rdfs:comment "The Global Positioning System (GPS) is a U.S.-owned utility
      ↪  that provides users with positioning, navigation, and timing (PNT)
      ↪  services. This system consists of three segments: the space segment,
      ↪  the control segment, and the user segment. The U.S. Space Force
      ↪  develops, maintains, and operates the space and control segments."@en ;
144   rdfs:isDefinedBy "https://www.gps.gov/systems/gps/"@en ;
145   rdfs:label "Global Positioning System"@en ;
146   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
147
148 ###  http://purl.org/poso/common/Galileo
149 :Galileo rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
150   rdfs:comment "Galileo is a global GNSS owned and operated by the European
      ↪  Union. The EU declared the start of Galileo Initial Services in 2016
      ↪  and plans to complete the system of 24+ satellites in 2021."@en ;
151   rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
152   rdfs:label "Galileo"@en ;
153   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
154
155 ###  http://purl.org/poso/common/GeolocationAPI
156 :GeolocationAPI rdf:type owl:NamedIndividual , poso:LocationBasedService ;
157   rdfs:comment "The Geolocation API is a W3C specification that provides
      ↪  geographical location data based on the capabilities of the hosting
      ↪  device."@en ;
158   rdfs:isDefinedBy "https://www.w3.org/TR/geolocation/"^^xsd:anyURI ;
159   rdfs:label "Geolocation API" .
160
161 ###  http://purl.org/poso/common/HectorSLAM
162 :HectorSLAM rdf:type owl:NamedIndividual , poso:VSLAM ;
163   rdfs:label "Hector SLAM"@en ;
164   rdfs:seeAlso "http://wiki.ros.org/hector_slam"^^xsd:anyURI .
165
166 ###  http://purl.org/poso/common/IRNSS
167 :IRNSS rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
168   owl:sameAs :NavIC ;
169   rdfs:comment "IRNSS is a regional GNSS owned and operated by the
      ↪  Government of India. IRNSS is an autonomous system designed to cover
      ↪  the Indian region and 1500 km around the Indian mainland. The system
      ↪  consists of 7 satellites. In 2016, India renamed IRNSS as the
      ↪  Navigation Indian Constellation (NavIC, meaning \"sailor\" or
      ↪  \"navigator\")."@en ;
170   rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
171   rdfs:label "Indian Regional Navigation Satellite System"@en ;
```

```
172   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
173
174 ###  http://purl.org/poso/common/IndoorAtlas
175 :IndoorAtlas rdf:type owl:NamedIndividual , poso:PositioningPlatform ;
176   rdfs:comment "Cross-platform indoor positioning SDK powered by patented
    ↪  6-layer sensor fusion core. Fast development cycles. Easy to set up.
    ↪  Proven by 10,000+ developers."@en ;
177   rdfs:isDefinedBy "https://www.indooratlas.com/"^^xsd:anyURI ;
178   rdfs:label "IndoorAtlas"@en .
179
180 ###  http://purl.org/poso/common/KNNFingerprinting
181 :KNNFingerprinting rdf:type owl:NamedIndividual , poso:Fingerprinting ;
182   rdfs:comment "k-Neirest Neighbour fingerprinting is a fingerprinting
    ↪  technique where a number (k) of matches are selected, as opposed to one
    ↪  single fingerprint match. The final output position is the (weighted)
    ↪  average of the k-number of selected fingerprints."@en ;
183   rdfs:label "k-NN fingerprinting"@en .
184
185 ###  http://purl.org/poso/common/LDPL
186 :LDPL rdf:type owl:NamedIndividual , poso:RadioPropagation ;
187   rdfs:comment "The log-distance path loss model is a radio propagation
    ↪  model that predicts the path loss a signal encounters inside a building
    ↪  or densely populated areas over distance."@en ;
188   rdfs:isDefinedBy
    ↪  <http://dbpedia.org/resource/Log-distance_path_loss_model> ;
189   rdfs:label "Log-distance path loss model"@en .
190
191 ###  http://purl.org/poso/common/MidpointLateration
192 :MidpointLateration rdf:type owl:NamedIndividual , poso:Lateration ;
193   rdfs:comment "Midpoint lateration uses the midpoint of two points."@en ;
194   rdfs:label "Midpoint lateration"@en .
195
196 ###  http://purl.org/poso/common/NLSMultilateration
197 :NLSMultilateration rdf:type owl:NamedIndividual , poso:Multilateration ;
198   rdfs:label "Non-linear least squares multilateration"@en .
199
200 ###  http://purl.org/poso/common/NavIC
201 :NavIC rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
202   rdfs:comment "IRNSS is a regional GNSS owned and operated by the
    ↪  Government of India. IRNSS is an autonomous system designed to cover
    ↪  the Indian region and 1500 km around the Indian mainland. The system
    ↪  consists of 7 satellites. In 2016, India renamed IRNSS as the
    ↪  Navigation Indian Constellation (NavIC, meaning \"sailor\" or
    ↪  \"navigator\")."@en ;
203   rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
204   rdfs:label "Navigation Indian Constellation"@en ;
205   <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
206
207 ###  http://purl.org/poso/common/ORBSLAM2
208 :ORBSLAM2 rdf:type owl:NamedIndividual , poso:VSLAM ;
209   rdfs:label "ORB-SLAM2"@en ;
210   rdfs:seeAlso "https://github.com/raulmur/ORB_SLAM2"^^xsd:anyURI .
211
```

```
212 ###  http://purl.org/poso/common/ORBSLAM3
213 :ORBSLAM3 rdf:type owl:NamedIndividual , poso:VSLAM ;
214   rdfs:isDefinedBy "https://github.com/UZ-SLAMLab/ORB_SLAM3"^^xsd:anyURI ;
215   rdfs:label "ORB-SLAM3"@en .
216
217 ###  http://purl.org/poso/common/OpenHPS
218 :OpenHPS rdf:type owl:NamedIndividual , poso:PositioningPlatform ;
219   rdfs:comment "OpenHPS is an open source hybrid positioning system to help
    ↪  developers fuse various positioning technologies and algorithms. The
    ↪  system offers a modular data processing framework with each modules
    ↪  ranging from computer vision to common algorithms such as
    ↪  fingerprinting or data persistence of sampled data."@en ;
220   rdfs:isDefinedBy "https://openhps.org"^^xsd:anyURI ;
221   rdfs:label "OpenHPS"@en .
222
223 ###  http://purl.org/poso/common/OpenVSLAM
224 :OpenVSLAM rdf:type owl:NamedIndividual , poso:VSLAM ;
225   rdfs:label "OpenVSLAM"@en ;
226   rdfs:seeAlso
    ↪  "https://stella-cv.readthedocs.io/en/latest/index.html"^^xsd:anyURI ;
227   <http://www.w3.org/2004/02/skos/core#altLabel> "stella-vslam"@en .
228
229 ###  http://purl.org/poso/common/ProbabilisticFingerprinting
230 :ProbabilisticFingerprinting rdf:type owl:NamedIndividual ,
    ↪  poso:Fingerprinting ;
231   rdfs:label "Probabilistic fingerprinting"@en .
232
233 ###  http://purl.org/poso/common/QZSS
234 :QZSS rdf:type owl:NamedIndividual , poso:SatellitePositioningSystem ;
235    rdfs:comment "QZSS is a regional GNSS owned by the Government of Japan
    ↪  and operated by QZS System Service Inc. (QSS). QZSS complements GPS to
    ↪  improve coverage in East Asia and Oceania. Japan declared the official
    ↪  start of QZSS services in 2018 with 4 operational satellites, and plans
    ↪  to expand the constellation to 7 satellites by 2023 for autonomous
    ↪  capability."@en ;
236    rdfs:isDefinedBy "https://www.gps.gov/systems/gnss/"^^xsd:anyURI ;
237    rdfs:label "Quasi-Zenith Satellite System"@en ;
238    <http://www.w3.org/ns/ssn/hasDeployment> :EarthDeployment .
239
240 ###  http://purl.org/poso/common/RFCellIdentification
241 :RFCellIdentification rdf:type owl:NamedIndividual ,
    ↪  poso:CellIdentification ;
242   rdfs:comment "RF cell identification is a technique that determines the
    ↪  position based on the closest RF landmark in range."@en ;
243   rdfs:label "RF cell identification"@en .
244
245 ###  http://purl.org/poso/common/ROS
246 :ROS rdf:type owl:NamedIndividual , poso:PositioningPlatform ;
247   rdfs:comment "ROS (Robot Operating System) provides libraries and tools
    ↪  to help software developers create robot applications. It provides
    ↪  hardware abstraction, device drivers, libraries, visualizers,
    ↪  message-passing, package management, and more."@en ;
248   rdfs:isDefinedBy "https://www.ros.org/"^^xsd:anyURI ;
```

```
249   rdfs:label "Robotics Operating System"@en .
250
251 ###  http://purl.org/poso/common/SVMFingerprinting
252 :SVMFingerprinting rdf:type owl:NamedIndividual , poso:Fingerprinting ;
253   rdfs:label "Support Vector Machine fingerprinting"@en .
254
255 ###  http://purl.org/poso/common/SonyNimway
256 :SonyNimway rdf:type owl:NamedIndividual , poso:PositioningPlatform ;
257   rdfs:comment "Nimway from Sony is a complete smart office solution for
    ↪  the people-centred workplace. Acting as your personal assistant
    ↪  throughout the workday, it facilitates many otherwise time-consuming
    ↪  tasks, like finding your way to a meeting room, locating a colleague or
    ↪  booking a desk."@en ;
258   rdfs:isDefinedBy "https://www.sonynetworkcom.com/nimway"^^xsd:anyURI ;
259   rdfs:label "Sony Nimway"@en .
260
261 ###  http://purl.org/poso/common/WeightedAccuracyFusion
262 :WeightedAccuracyFusion rdf:type owl:NamedIndividual, poso:HighLevelFusion;
263   rdfs:comment "Decision level fusion based on weighted average of the
    ↪  accuracy^-1."@en ;
264   rdfs:label "Weighted accuracy fusion"@en .
```

Listing B.1: POSO common positioning systems and algorithms

B.2.2 Crowdsourcing via Solid (WiP)

Setting up an indoor positioning system is a complicated, time-intensive task and often requires calibration with different hardware. Crowdsourcing is a common practice in the set-up of indoor positioning systems and is one of the foundational features of some of our related work such as AnyPlace [160]. However, one of the limitations of crowdsourcing is the ownership of data and the transparency of which data is being collected. Furthermore, crowdsourcing also entails that the aggregator should get some validation on the data's validity to prevent the calibration from containing data that could negatively influence the localisation accuracy.

To solve this issue, we started with the design of an extension for the POSO ontology that would facilitate this crowdsourcing. While the extension and accompanying OpenHPS module were never finished, we envisioned that positioning systems would have a set of *procedures* to indicate the steps the user must take to start calibration. These procedures contained actions such as "move forward", "rotate 90 degrees" or an action to input a position. A user would then collect sensor data specified by the procedure and store this data in their Solid Pod. After the collection, the user could then share this data with the positioning system. In general, the idea of the `poso-crowdsource` extension was to offer a descriptive vocabulary capable of describing all system and user procedures involved with crowdsourcing.

```
@prefix : <http://purl.org/poso/crowdsource/> .

# ...

#################################################################
#    Classes
#################################################################

###  http://purl.org/poso/crowdsource/CrowdsourceSystem
:CrowdsourceSystem rdf:type owl:Class ;
  rdfs:subClassOf <http://www.w3.org/ns/ssn/System> ;
  rdfs:label "Crowdsource system"@en ;
  skos:altLabel "Crowdsourcing system"@en .

###  http://purl.org/poso/crowdsource/Device
:Device rdf:type owl:Class ;
  rdfs:subClassOf <http://www.w3.org/ns/sosa/FeatureOfInterest> ,
          <http://www.w3.org/ns/ssn/System> .

###  http://purl.org/poso/crowdsource/InputAction
:InputAction rdf:type owl:Class ;
  rdfs:subClassOf :UserAction ;
  rdfs:comment "A user input action is a procedure where a user inputs
↪  information."@en ;
  rdfs:label "User input action"@en .

###  http://purl.org/poso/crowdsource/InputOrientationAction
:InputOrientationAction rdf:type owl:Class ;
  rdfs:subClassOf :InputAction ;
  rdfs:comment "A user input orientation action is a procedure where a user
↪  inputs their current orientation."@en ;
  rdfs:label "User input orientation action"@en .

###  http://purl.org/poso/crowdsource/InputPositionAction
:InputPositionAction rdf:type owl:Class ;
  rdfs:subClassOf :InputAction ;
  rdfs:comment "A user input orientation action is a procedure where a user
↪  inputs their current position."@en ;
  rdfs:label "User input position action"@en .

###  http://purl.org/poso/crowdsource/ObserveAction
:ObserveAction rdf:type owl:Class ;
  rdfs:subClassOf :SystemAction ;
  rdfs:label "Observe action"@en .

###  http://purl.org/poso/crowdsource/SystemAction
:SystemAction rdf:type owl:Class ;
  rdfs:subClassOf <http://www.w3.org/ns/sosa/Procedure> ,
    [ rdf:type owl:Restriction ;
  owl:onProperty <http://www.w3.org/ns/ssn/implementedBy> ;
  owl:allValuesFrom :CrowdsourceSystem ] ;
    rdfs:comment "A system action is a procedure executed by a system."@en
↪  ;
```

```
50      rdfs:label "System action"@en .
51
52  ###  http://purl.org/poso/crowdsource/UserAction
53  :UserAction rdf:type owl:Class ;
54    rdfs:subClassOf <http://www.w3.org/ns/sosa/Procedure> ,
55      [ rdf:type owl:Restriction ;
56        owl:onProperty <http://www.w3.org/ns/ssn/implementedBy> ;
57        owl:allValuesFrom :CrowdsourceSystem ] ;
58    rdfs:comment "A user action is a procedure executed by the user."@en ;
59    rdfs:label "User action"@en .
60
61  ###  http://purl.org/poso/crowdsource/UserMoveAction
62  :UserMoveAction rdf:type owl:Class ;
63    rdfs:subClassOf :UserAction ;
64    rdfs:comment "A user move action is a procedure where a user is asked to
    ↪  move to a certain location."@en ;
65    rdfs:label "User move action"@en .
66
67  ###  http://purl.org/poso/crowdsource/UserRotateAction
68  :UserRotateAction rdf:type owl:Class ;
69    rdfs:subClassOf :UserAction ;
70    rdfs:comment "A user move action is a procedure where a user is asked to
    ↪  rotate to a certain orientation."@en ;
71    rdfs:label "User rotate action"@en .
72
73  ###  http://purl.org/poso/crowdsource/UserWaitAction
74  :UserWaitAction rdf:type owl:Class ;
75    rdfs:subClassOf :UserAction ;
76    rdfs:comment "A user move action is a procedure where a user is asked to
    ↪  wait."@en ;
77    rdfs:label "User wait action"@en ;
78    skos:example "A user wait action can be combined with a system action to
    ↪  wait while the system finishes a certain procedure."@en .
```

Listing B.2: POSO extension to describe crowdsourcing instructions (Work in Progress)

B.2.3 FidMark

This appendix contains information about the current state of the Fiducial Marker ontology (FidMark) that extends the POSO ontology. Figure B.2 shows the main classes that FidMark extends from the POSO ontology. We make a clear classification of active and passive markers [257] as this represents the major difference between different types of markers. We further classified passive markers into barcodes that can encode information such as an identifier. In addition to these main classes, we currently provide over 30 different types of markers as subclasses of the fiducial marker class[1], such as reacTIVision [258] or CCTag [259].

[1]For more information about the ontology profile and ontology statistics, please check the online documentation.

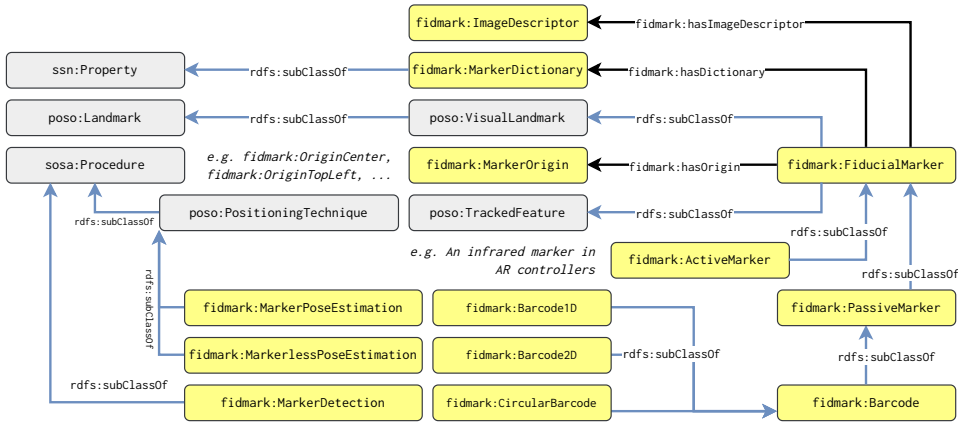


Figure B.2: FidMark main classes

With our FidMark ontology, we aim to describe different markers, their setup and their position relative to a certain reference space. The basic architecture and a use case for our ontology are demonstrated in Figure B.3. We describe markers, their identifier, position and orientation using the combination of FidMark and POSO. FidMark handles the description of the markers while POSO is used to describe the markers' absolute and relative position as visual landmarks. AR-capable devices with access to this description can synchronise their reference frame due to the common description. Any virtual object placed relative to these markers can again use the generality of POSO to indicate its relative position.

In our ontology, each type of fiducial marker is annotated with additional metadata describing the markers' visual and functional properties such as their shape, colour and encoding method. When available in OpenCV [186], we also provide the dictionary names of markers as they are available in OpenCV to facilitate the development.

To demonstrate the generality of FidMark for applications outside the domain of augmented reality, we provide the `fidmark-dicom` alignment ontology to align fiducial markers and POSO with the Digital Imaging and Communications in Medicine (DICOM) ontology [260] in our supplemental material.

Fiducial markers have a set of properties that are sub-properties of properties on `sosa:FeatureOfInterest` from the Sensors, Observations, Systems and Actuators (SOSA) ontology [131]. They define how the marker should be detected, decoded and mapped to its location within the physical space. With fiducial markers being used to determine the relative translation, scale or orientation of visual objects within the field of view. The object and data properties we added to our ontology are also chosen to enable this description.

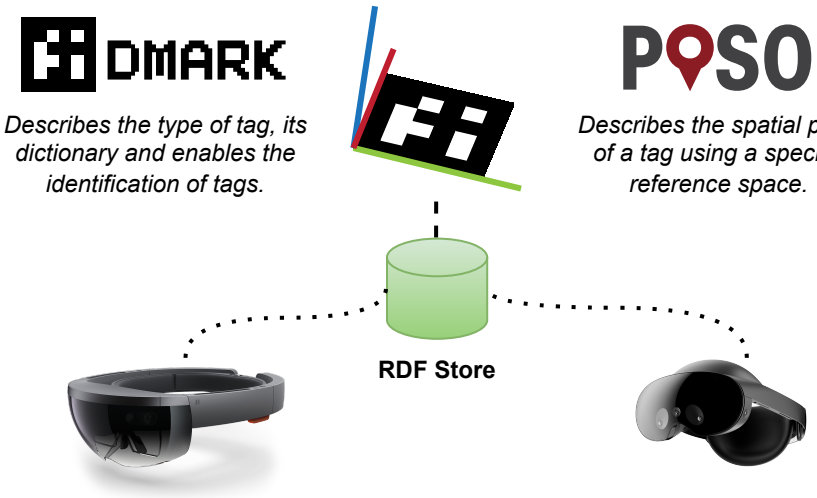


Figure B.3: Basic usage of FidMark together with the POSO ontology

We provide the `:hasOrigin` predicate and `:MarkerOrigin` property to indicate the position origin on the marker itself. This origin determines the 2D or 3D location on the fiducial marker. An origin description is required to determine the accurate translation when performing relative positioning from virtual objects relative to the marker.

- **dictionary**: A set of marker identifiers that are available using a specified encoding scheme. A dictionary is also referred to as a *marker symbology* or *marker family* [261]. This encoding scheme can be generalised to both binary encoding in barcodes as well as the encoding with active markers such as infrared markers [262].
- **origin**: The origin of the marker is an important design requirement in order to determine the relative orientation and translation to the marker. Our `:hasOrigin` predicate and `:MarkerOrigin` property are based on the OpenCV *pattern*.
- **dimensions**: The known dimensions of a marker can be used to determine its scale. Our ontology supports the specification of both, a marker's width and height to support rectangular markers such as AprilTag [263].
- **hamming distance**: The minimum hamming distance between two codes, represented by the minimum number of bits that must be changed in one tag's code to reach another tag's code.
- **image descriptor**: The marker image descriptor links to an image URI or a Base64 representation of the image. Alternatively, the image descriptor can

be described as a processed descriptor for natural feature tracking [264] as illustrated in Figure B.4.

- **identifier**: A numeric identifier that can uniquely identify a marker from a (pre-)defined *dictionary*. The identifier is (part of) the encoded data.
- **data**: Other than an identifier, some marker types allow the encoding of other types of binary data. An example of such a marker is a QR code, which can be used for both, pose estimation as well as the encoding of binary data.
- **codes & marker bits**: Frameworks with no prior knowledge of the concept of a dictionary or how it is computed require a list of all available codes that can be encoded with the available `:markerBits` and error correction.

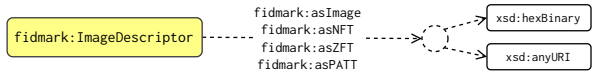


Figure B.4: Image descriptor usage for markers

For a complete list of properties, please refer to the online documentation in our supplemental material also provides marker-specific object and data properties[2].

[2] `https://openhps.github.io/FidMark/1.0/en/`

Appendix C

SemBeacon

C.1 Specification 1.0

This section contains information about the first version of the SemBeacon specification. The latest version can be found in Section 5.2.

Bit	Description
0	Indicates if beacon has a position (0 = unsure, 1 = yes).
1	Indicates if beacon is private (0 = public, 1 = private).
2	Indicates if beacon is stationary (0 = stationary, 1 = mobile).
3	Indicates if the beacon is part of a positioning system (0 = no, 1 = yes).
4	Indicates if beacon provides telemetry data (0 = no, 1 = yes).
5–7	Reserved for future use.

Table C.1: SemBeacon v1.0 flags

Table C.1 showcases the first version of the flags in the SemBeacon specification. Version 1.1 did not introduce any breaking changes to these flags. However, terminologies were changed which expanded the use cases of some of the flags.

C.2 Arduino Library

An Arduino library is available which facilitates the broadcasting and scanning of a SemBeacon on both BLE v4 and BLE v5 devices. The library only handles the broadcasting on embedded devices and does not retrieve any linked data from the URIs. The library is available via the Arduino IDE:
`https://docs.arduino.cc/libraries/esp32_sembeacon/`

```
#define GPIO_DEEP_SLEEP_DURATION 100 // Sleep duration in ms
#define BLE_ADVERTISEMENT_INTERVAL 1
#define BEACON_UUID "77f340db-ac0d-20e8-aa3a-f656a29f236c"

/* Uncomment the following two lines to disable BLE5 */
#include "sdkconfig.h"
#undef CONFIG_BT_BLE_50_FEATURES_SUPPORTED
```

```
8  /* ------------------------------------------------ */
9
10 #include <BLESemBeacon.h>
11 #include <BLESemBeaconAdvertising.h>
12
13 #include "BLEDevice.h"
14 #include "esp_sleep.h"
15
16 BLESemBeaconAdvertising *advertising;
17
18 void createBeacon() {
19   BLESemBeacon beacon = BLESemBeacon();
20   // Choose 0x004C for Apple Ltd. Note that this will
21   // not automatically be detected on iOS
22   beacon.setManufacturerId(0xFFFF);
23   // RSSI at 1m distance
24   beacon.setSignalPower(-30);
25   // Namespace UUID
26   beacon.setNamespaceId(BLEUUID(BEACON_UUID));
27   // Instance Identifier
28   beacon.setInstanceId(0x9c7ce6fc);
29   // URI to the resource
30   beacon.setResourceURI("https://bit.ly/3JsEnF9");
31   beacon.setBeaconFlags(
32     SEMBEACON_FLAG_HAS_POSITION | SEMBEACON_FLAG_HAS_SYSTEM
33   );
34
35   advertising->setBeacon(&beacon);
36 }
37
38 void setup() {
39   // Create the BLE Device
40   BLEDevice::init("");
41   advertising = new BLESemBeaconAdvertising();
42   // Create the beacon data
43   createBeacon();
44 }
45
46 void loop() {
47   // Start advertising
48   advertising->start();
49   delay(100);
50   advertising->stop();
51   esp_deep_sleep(1000LL * GPIO_DEEP_SLEEP_DURATION);
52 }
```

Listing C.1: Example usage of the SemBeacon Arduino library

Listing C.1 shows an example of the `ESP32_SemBeacon` Arduino library. This particular example is used to create the beacons in our proof of concept application explained in Section 6.6.

The use of the library is similar to the existing BLE library for ESP32[1] with our classes expanding upon this library. Developers can create a SemBeacon by specifying the basic information (i.e., the manufacturer and signal power) and then providing a namespace and instance identifier. Next, a (shortened) URL is provided in combination with a set of flags. Depending on the availability of BLE v5, the library will automatically determine if the SemBeacon should be broadcasted as a BLE v5 or v4 beacon.

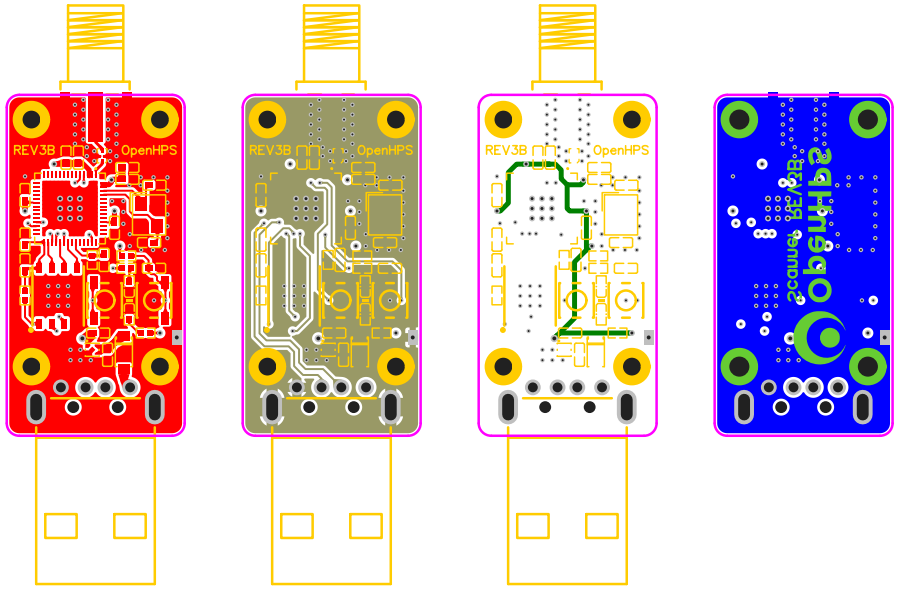
C.3 Hardware

For evaluating our SemBeacons, we developed our own hardware which included an ESP32-S3 microcontroller that supports Bluetooth 5.0. At the time of the development, no development boards were available in Europe. Four revisions of the hardware were made. These revisions mainly fixed issues concerning the antenna and RF interference. Our final revision uses a 4 layer PCB design with an external SMA antenna and USB-A port.

Originally, the hardware was designed as a Bluetooth and Wi-Fi scanner, similar to our earlier work [99] where we designed battery-powered scanners to perform indoor navigation. While the hardware still supports this functionality, we focused on using the hardware to broadcast SemBeacons.

[1] https://github.com/espressif/arduino-esp32/tree/master/libraries/BLE

C.3.1 PCB



(a) Top layer *(b)* Inner layer 1 *(c)* Inner layer 2 *(d)* Bottom layer

Figure C.1: PCB layers of the OpenHPS ESP32 scanner revision 3B

C.3.2 Schematic

The design of the ESP32-S3 hardware was based on the FeatherS3[2] schematic. However, with our design, we wanted to leverage the OTG USB support, increase the amount of flash memory to enable a small website to run on the device(s) and also use an external antenna. Figure C.2 shows the final schematic of our hardware.

[2] https://github.com/UnexpectedMaker/esp32s3/blob/main/schematics/schematic-feathers3_p7.pdf

C.3.3 Bill of Material

#	Manufacturer Part	Manufacturer	Supplier	Supplier Part	Price
1	W25Q128JVPIQ	WINBOND	LCSC	C2803379	1.405
1	USB-05	SOFNG	LCSC	C112454	0.098
1	HJ-SMA023		LCSC	C1509219	0.615
1	SLFL15-2R450G-01TF	Sunlord	LCSC	C96778	0.032
1	ESP32-S3	ESPRESSIF	LCSC	C2913192	2.698
2	0402WGF680JTCE	UniOhm	LCSC	C25131	0.001
1	FC-F1005UGK-520M5	NATIONSTAR	LCSC	C130723	0.037
2	0402CG220J500NT	FH	LCSC	C1555	0.001
1	NCP167AMX330TBG	ON Semiconductor	LCSC	C603662	1.403
4	CC0402KRX7R7BB104	YAGEO	LCSC	C60474	0.001
7	CL05A105KP5NNNC	SAMSUNG	LCSC	C14445	0.002
1	TZ0475B	TST	LCSC	C501793	0.139
2	TS-1177-C-B-B	XKB Enterprise	LCSC	C561512	0.06
1	0402B103K500NT	FH	LCSC	C1524	
1	B5819W	MDD	LCSC	C64885	0.018
1	RC0402FR-07100KL	YAGEO	LCSC	C60491	
1	0402WGF4423TCE	UniOhm	LCSC	C5101	0.001
1	0402WGF1002TCE	UniOhm	LCSC	C25744	0.001
1	FC-F1005HRK-620H5	NATIONSTAR	LCSC	C130719	0.034

Table C.2: Bill of material (prices in USD)

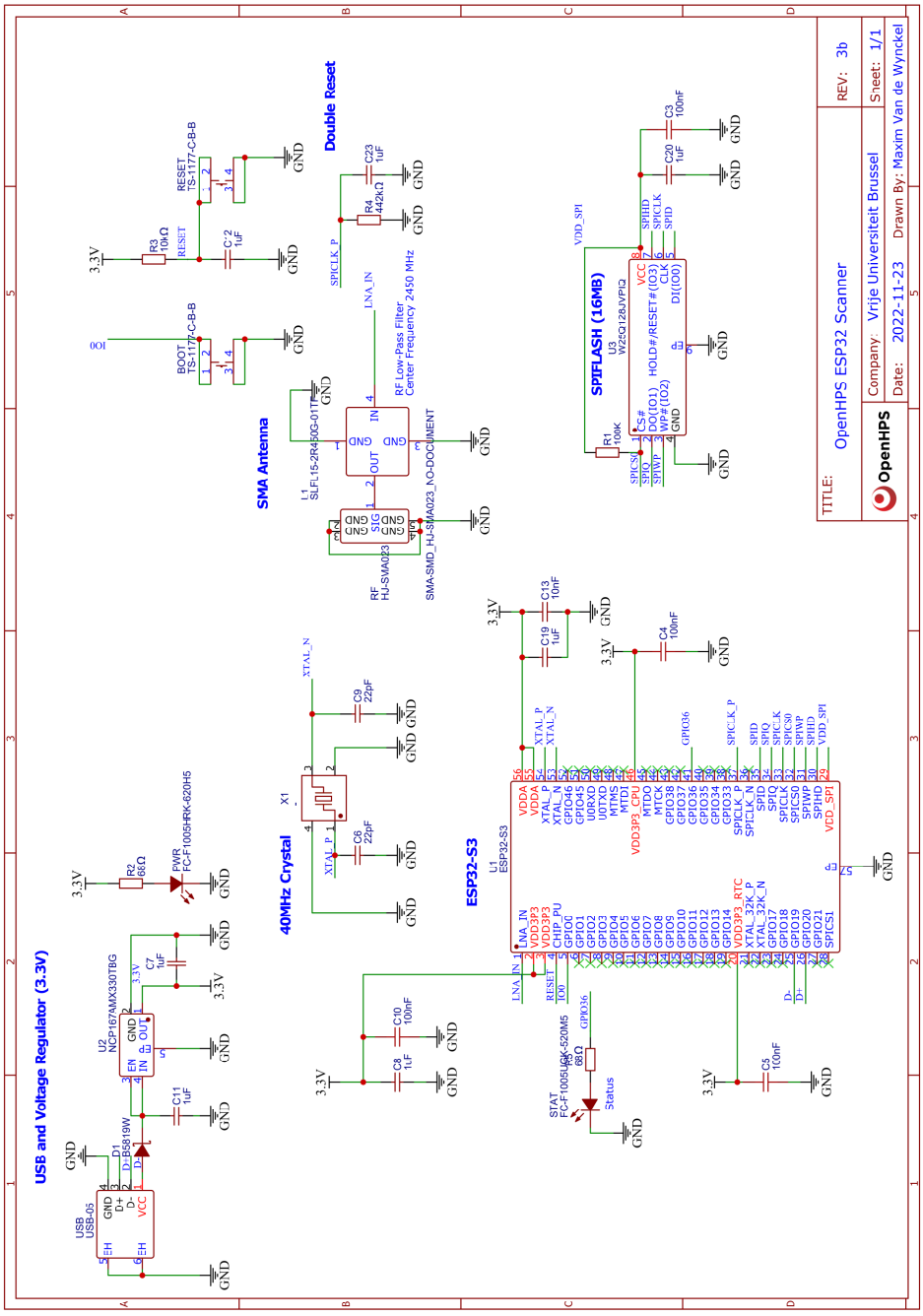


Figure C.2: Schematic of OpenHPS ESP32 Scanner

C.4 Ontology

SemBeacon comes with a small vocabulary extension to the Positioning System Ontology (POSO) that adds `SemBeacon` as a type of `BluetoothBeacon` with a namespace and instance identifier.

```
#################################################################
#  Object Properties
#################################################################

###  http://purl.org/sembeacon/namespace
:namespace rdf:type owl:ObjectProperty ;
  rdfs:domain <http://purl.org/poso/BluetoothBeacon> ;
  rdfs:range <http://www.w3.org/ns/ssn/Deployment> ;
  rdfs:comment "The namespace property directs to the deployment containing
↪  all sensors deployed in this namespace."@en ;
  rdfs:label "namespace"@en .

#################################################################
#  Data properties
#################################################################

###  http://purl.org/sembeacon/instanceId
:instanceId rdf:type owl:DatatypeProperty ;
   rdfs:domain :SemBeacon ;
   rdfs:range xsd:hexBinary ,
     [ rdf:type rdfs:Datatype ;
       owl:onDatatype xsd:hexBinary ;
       owl:withRestrictions ( [ xsd:maxLength "8"^^xsd:int ] )
     ] ;
   rdfs:comment "An instance identifier is the 32-bit UUID that defines the
↪  instance of a SemBeacon within a namespace."@en ;
   rdfs:label "Instance ID"@en .

###  http://purl.org/sembeacon/namespaceId
:namespaceId rdf:type owl:DatatypeProperty ;
    rdfs:domain <http://purl.org/poso/BluetoothBeacon> ,
       <http://www.w3.org/ns/ssn/Deployment> ;
    rdfs:range [ rdf:type rdfs:Datatype ;
      owl:onDatatype xsd:hexBinary ;
      owl:withRestrictions ( [ xsd:maxLength "32"^^xsd:int ] )
    ] ;
    rdfs:comment "A namespace identifier is the 128-bit UUID that defines
↪  the namespace of a SemBeacon."@en ;
    rdfs:label "Namespace ID"@en .

###  http://purl.org/sembeacon/shortResourceURI
:shortResourceURI rdf:type owl:DatatypeProperty ;
     rdfs:domain :SemBeacon ;
     rdfs:range xsd:anyURI ;
```

```
42      rdfs:comment "Shortened resource URI. The resource URI should resolve
    ↪  to the full resource URI of the resource that this predicate is used
    ↪  in."@en ;
43      rdfs:label "Short resource URI"@en ;
44      <http://www.w3.org/2004/02/skos/core#example>
    ↪  "https://tinyurl.com/5jxncuvy"@en .
45
46  ###  http://purl.org/sembeacon/version
47  :version rdf:type owl:DatatypeProperty ;
48      rdfs:domain :SemBeacon ;
49      rdfs:range xsd:string ;
50      rdfs:comment "SemBeacon version"@en ;
51      rdfs:label "version"@en ;
52      <http://www.w3.org/2004/02/skos/core#example> "1.1"@en .
53
54  ###################################################################
55  #  Classes
56  ###################################################################
57
58  ###  http://purl.org/poso/BluetoothBeacon
59  <http://purl.org/poso/BluetoothBeacon> rdf:type owl:Class .
60
61  ###  http://purl.org/sembeacon/SemBeacon
62  :SemBeacon rdf:type owl:Class ;
63    rdfs:subClassOf <http://purl.org/poso/BluetoothBeacon> ,
64          [ rdf:type owl:Restriction ;
65          owl:onProperty <http://purl.org/poso/common/hasReferenceRSSI> ;
66          owl:minCardinality "1"^^xsd:nonNegativeInteger
67          ] ;
68    rdfs:comment "SemBeacon is a semantic beacon that broadcasts an URI
    ↪  describing its position and references its deployment."@en ;
69    rdfs:label "SemBeacon"@en ;
70    rdfs:seeAlso <http://purl.org/poso/common/SemBeacon> .
```

Listing C.2: SemBeacon ontology extensions of POSO

Appendix D

Linked Data Hash Tree Specification

```
@prefix : <http://purl.org/ldht/> .

# ...

#################################################################
#    Object Properties
#################################################################

###  http://purl.org/ldht/hashFunction
:hashFunction rdf:type owl:ObjectProperty ;
  rdfs:comment "The hashing function describes how input data is
  ↪  transformed to a value"@en ;
  rdfs:label "Hashing function"@en .

###  http://purl.org/ldht/key
:key rdf:type owl:ObjectProperty ;
  rdfs:domain :Entry ;
  rdfs:label "key"@en .

###  http://purl.org/ldht/nodes
:nodes rdf:type owl:ObjectProperty ;
  rdfs:subPropertyOf owl:topObjectProperty ;
  rdfs:domain :Collection ;
  rdfs:range <https://w3id.org/tree#Collection> ;
  rdfs:comment "Indicates the collection of nodes for a particular linked
  ↪  data hash table"@en ;
  rdfs:label "Nodes collection"@en .

###  http://purl.org/ldht/value
:value rdf:type owl:ObjectProperty ;
  rdfs:domain :Entry ;
  rdfs:label "value"@en .

#################################################################
#    Data properties
#################################################################

```

```
36 ###  http://purl.org/ldht/nodeID
37 :nodeID rdf:type owl:DatatypeProperty ;
38   rdfs:domain :Node ;
39   rdfs:range xsd:integer ;
40   rdfs:comment "The node identifier is a number generated by using the hash
     ↪  algorithm of the network."@en ;
41   rdfs:label "node identifier"@en .
42
43 ###  http://purl.org/ldht/timeout
44 :timeout rdf:type owl:DatatypeProperty ;
45   rdfs:domain schema:Action ;
46   rdfs:range xsd:integer ;
47   rdfs:comment "The timeout is the time in milliseconds that a node will
     ↪  wait for a response from another node."@en ;
48   rdfs:label "timeout"@en .
49
50 ###################################################################
51 #    Classes
52 ###################################################################
53
54 ###  http://purl.org/ldht/AddNodeAction
55 :AddNodeAction rdf:type owl:Class ;
56   rdfs:subClassOf schema:AddAction ;
57   rdfs:comment "The act of indicating to a node that a new node has joined
     ↪  the network."@en ;
58   rdfs:label "Add node action"@en .
59
60 ###  http://purl.org/ldht/Collection
61 :Collection rdf:type owl:Class ;
62   rdfs:subClassOf <https://w3id.org/tree#Collection> ;
63   rdfs:comment "A ldht:Collection is a tree collection."@en ;
64   rdfs:label "Collection"@en .
65
66 ###  http://purl.org/ldht/Entry
67 :Entry rdf:type owl:Class ;
68   rdfs:label "Entry"@en .
69
70 ###  http://purl.org/ldht/Node
71 :Node rdf:type owl:Class ;
72   rdfs:subClassOf <https://w3id.org/tree#Node> ;
73   rdfs:comment "A ldht:Node is a node in a distributed hash tree that may
     ↪  contain relations to other nodes in the distributed hash tree."@en ;
74   rdfs:label "Node"@en .
75
76 ###  http://purl.org/ldht/PingAction
77 :PingAction rdf:type owl:Class ;
78   rdfs:subClassOf schema:Action ;
79   rdfs:comment "The act of pinging a node to request if it is still active
     ↪  and responding to actions."@en ;
80   rdfs:label "Ping action"@en .
81
82 ###  http://purl.org/ldht/RemoveNodeAction
83 :RemoveNodeAction rdf:type owl:Class ;
```

```
84    rdfs:subClassOf schema:DeleteAction ;
85    rdfs:comment "The act of indicating to a node that another node is
      ↪  removed in the network."@en ;
86    rdfs:label "Remove node action"@en .
87
88
89  ###  http://purl.org/ldht/StoreValueAction
90  :StoreValueAction rdf:type owl:Class ;
91    rdfs:subClassOf schema:InsertAction ;
92    rdfs:comment "The act of storing a value in a node."@en ;
93    rdfs:label "Store value action"@en .
```

Appendix E

Survey on the Privacy and Transparency of Location Data

This appendix contains information on our survey on the "Privacy and Transparency of Location data". The survey was launched in January 2025 and was primarily used to determine how users perceive the privacy and transparency of their location data in mobile applications. Our complete pseudonymised results were published on Zenodo [4], which also includes screenshots of the questions.

E.1 Questions

In this section, we list all the questions that were asked in the survey. We also provide additional context as to why some of these questions were asked. Each question is numbered[1] (e.g., Q5), this is an internal numbering and corresponds to the results presented in Section E.2.

E.1.1 General Awareness

We first started with general questions concerning their awareness of location data privacy. Some of these questions also serve as redundant questions for other questions. The possible answers to these questions are sometimes intentionally vague or self-explanatory to encourage participants to think critically about the potential impacts. Additionally, these questions also serve to scope the follow-up questions.

(Q2) How familiar are you with the concept of location data tracking?

In this question, participants were asked about their familiarity with location data tracking. Choices ranged from *"Not at all familiar"* (1) to *"Extermely familiar"* (5).

[1]Numbering starts at Q2 due to the first question being the GDPR consent

(Q3) How can services or companies track your physical location? (Check all that apply)

The following options were presented in a random order:

1. By sharing my location data using an application
2. Through surveillance cameras
3. By monitoring wireless activity
4. By accessing cloud data
5. Through my visits to websites
6. By intercepting cellular signals
7. Through my social media posts

Theoretically, most options apply, but the granularity of the location differs. This question was aimed to assess the general awareness of which types of interactions and technologies could be used to ensure that the participant thinks critically.

(Q4) Which of the following applications do you think use your location data? (Check all that apply)

The following types of applications were presented in a random order:

1. GPS/Maps applications (e.g., Google Maps, Waze)
2. Social media apps (e.g., Facebook, Instagram)
3. Weather apps
4. E-commerce platforms (e.g., Amazon, eBay)
5. Ride-sharing or food delivery apps
6. Smart home devices or assistants (e.g., Alexa, Google Home)
7. Travel or airline apps
8. Fitness or health-tracking apps (e.g., Fitbit, Strava)
9. Banking apps
10. E-mail clients
11. Applications to control smart devices
12. None of the above

(Q5) Have you ever installed a navigation application for one specific building?

This question was given with additional context: *For navigating an airport, conference, hospital or any other building*. Participants could answer yes, no or unsure.

(Q6) How often do you check the permissions granted to apps regarding location tracking?

Participants could choose between always, often, sometimes, rarely or never.

(Q7) How often do you read privacy policies when downloading or using apps?

The question was posed as a gauge from 0 (never) to 10 (always).

(Q8) Without checking, what percentage of applications on your smartphone have constant access to your location data?

This question was given with additional context: *Constant access entails that an application can request your current location at any time even when the application is not running*. The question was posed as a gauge from 0% to 100%.

(Q9) On a scale from 1 to 10, how valuable do you think your location data is to third parties?

The question was posed as a gauge from 1 to 10.

E.1.2 Privacy Concerns

This set of questions aims to asses any privacy issues associated with location data.

(Q10) Rank the following concerns about location data tracking from most concerning to least concerning.

1. Data being shared with third parties
2. Data being used for targeted advertising
3. Data being stolen in a breach
4. Lack of transparency about data usage
5. Being monitored by authorities or employers
6. Being monitored by friends or family
7. Being denied access to services due to location-based profiling

8. Applications collecting more information than required
9. Unclear consent processes (e.g., not knowing what you are agreeing to)
10. The ability of third parties to track you across different devices

(Q11) How concerned are you about the following scenarios involving your location data?

Participants were given five scenarios. Each scenario could be ranked on a Likert scale from "not at all concerned" (1) to "extremely concerned" (5).

1. Your location data being sold to third parties without your consent.
2. Your location data being used to create a detailed profile about you.
3. Being denied access to services due to location-based profiling.
4. Your location being used to track your movements without your knowledge.
5. Your location data being vulnerable to hacking or breaches.

(Q12) Do you feel you have enough control over how your location data is used?

Likert scale from "definitely not" (1) to "definitely yes" (5). When users indicated that they definitely did not have enough control, they were asked a follow-up question: *[Q13] Why do you feel that you need more control over how your location data is used?* Users who indicated that they definitely have enough control were asked the opposite: *[Q14] Why do you feel that you have enough control over how your location data is used?*

(Q15) Have you ever refused to use an app or service because of privacy concerns about location data?

A yes or no question asking if users ever refused to use an application due to privacy concerns.

(Q16) What actions, if any, do you take to limit location tracking?

An open question aimed to learn more about any actions participants take to protect their location data.

E.1.3 Transparency of Applications and Systems

This set of questions is aimed to determine how users perceive the transparency on the user of their location data. Even if an application or service provides a privacy

policy, users are often unaware of the implications when rejecting this policy or what data is shared.

(Q17) How transparent do you believe apps and systems are about how they use location data?

Participants were given five statements. For each statement, they could indicate their level of agreement on a Likert scale from 1 (strongly disagree) to 5 (strongly agree).

1. Apps clearly explain why they need my location data.
2. Privacy policies are easy to understand.
3. I feel informed about how my location data is shared.
4. I feel informed on how often an application can access my location data.
5. I am aware of the impact when I decline sharing my location data with an app.

(Q18) Would you trust an app more if it provided real-time notifications or visual indicators when it accesses your location data?

This question was given with additional context: *A visual indicator is similar to how modern smartphones show an icon to indicate that the camera is used.*

Three options were available:

1. Yes, I would feel much more confident.
2. Yes, but only if the notifications or visual indicators were not intrusive.
3. No, notifications or visual indicators would not change my level of trust.

(Q19) If a company was found to be misusing location data, how would it affect your behaviour?

Five choices were presented. These choices were determined based on the possible scenarios that a user could take after discovering data misuse, these scenarios are based on the topics of the European Data Protection Supervisor (EDPS)[2].

1. I would stop using their services entirely.
2. I would review and change my data-sharing permissions.
3. I would switch to a competitor with better transparency.

[2] https://www.edps.europa.eu/

4. I would demand more accountability from the company.
5. It wouldn't affect my behaviour.

(Q20) What's your preferred way for apps to communicate their location data policies?

1. A clear and concise summary during setup
2. A detailed privacy policy document
3. A short video explaining data practices
4. Periodic notifications about data usage
5. Other (please specify below)
   Participants were given a text field to specify, identified in the pseudonymised results as Q21_TEXT

(Q21) How comfortable would you be if apps stored your location data directly on your device or in a private space you control, instead of on their servers?

Likert scale from "extremely uncomfortable" (1) to "extremely comfortable" (5).

E.1.4 Valuation of Location Data

With this line of questions, we wanted to determine how valuable users find their location data. We used a relative scale to assess the valuation of location data by asking participants how they rank their location data compared to other personal data.

(Q22) How important is location data to the functionality of the apps you use daily?

Participants could choose between; not at all important, slightly important, moderately important, very important, extremely important, and "I don't know".

(Q23) Would you pay for an app or service that is guaranteed not to track your location?

This question was given with additional context: *Example: If a (free) service mentions that they use your location data for targeted advertising, but you can disable it by paying a fee*. Three options were given; "yes, definitely", "maybe, depending on the cost" or "no, I would not pay".

(Q24) If an app or system requests location data, what kind of benefit or incentive would justify this for you? (Check all that apply)

1. Improved app performance or features
2. Personalised content or offers
3. Monetary rewards (e.g., discounts, cashback)
4. None, I prefer not to share my location data
5. Other (please specify below)
   Participants were given a text field to specify, identified in the pseudonymised results as Q25_TEXT

(Q25) Why do you think companies collect location data?

1. To improve app features and functionality
2. For targeted advertising and marketing
3. To sell data to third parties
4. For analytics and business insights
5. To comply with regulatory or safety requirements
6. To improve the user experience
7. Other (please specify below)
   Participants were given a text field to specify, identified in the pseudonymised results as Q26_TEXT

(Q26) For each type of tracking, indicate how valuable you think it is for third parties

This question was given with additional context: *A third party is a company that purchases location data from a source that collects it*. Participants were given five examples of tracking that they could rank on a scale from "not valuable at all" (1) to "extremely valuable" (5).

1. A current snapshot of your exact location (e.g., where you are now)
2. Constant tracking of your movements (e.g., over days or weeks)
3. An estimate of your general location (e.g., city level)
4. Past location history (e.g., where you've been in the last month)
5. Aggregated location trends (e.g., patterns derived from multiple users)

(Q27) Rank the following personal data in order of importance to you (important to less important)

With this question, we wanted to determine how users rank their location data compared to other personal information. The following types of data were listed in random order, participants could move the items to rank them. The types of data were loosely based on the definitions for Personally Identifiable Information (PII) in the General Data Protection Regulation (GDPR) [9].

1. Location data
2. Web browsing activity
3. Home address
4. Phone number
5. E-mail address
6. Social media profile data
7. Health and/or medical information
8. Private calendar information
9. Photos
10. E-mail messages

E.1.5 Demographic Information

For privacy reasons and also because more specific user information was not required for the scope of this survey, our demographic questions were limited to age (Q28) and an indication of whether or not the participant lived inside the EU (Q29). The latter was used to determine if existing regulations such as the GDPR apply to this participant, which can influence the choices made in regard to transparency.

E.2 Pseudonymised Results

In total, 58 users participated in the survey. On average, participants spent 13 minutes on the survey with a median time of 12 minutes. Results were processed to remove identifiable information such as the IP address, rough estimate of the location, start and end time and internal identifiers used by Qualtrics. Next, unnecessary information was removed from the results, which is only used to determine the validity of the responses (e.g., captcha results and progress). Responses were originally sorted based on the start time but were randomised for the pseudonymised result.

After this processing, the open answers were analysed to determine if any identifiable information was present. The open responses were cleaned to remove empty answers or invalid characters. Finally, open answers in Dutch (i.e., one of the languages in which the survey was distributed) were translated into English. This translation is explicitly mentioned in the results through a separate column.

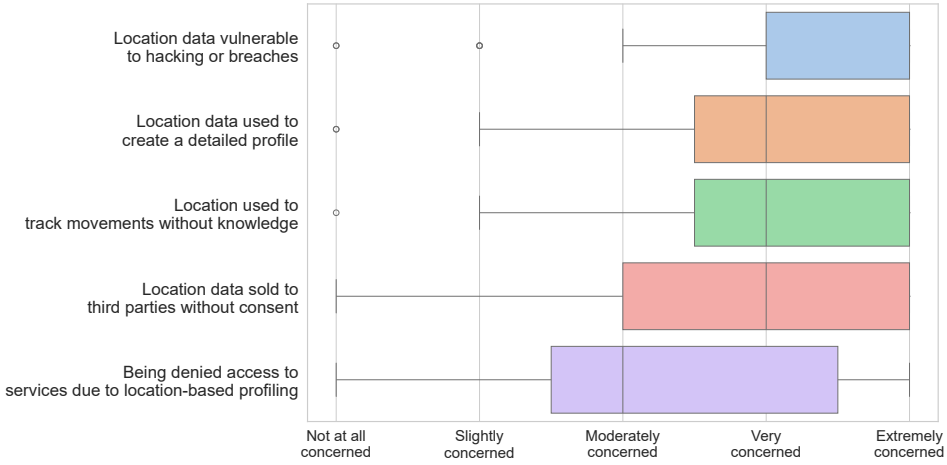


Figure E.1: User concerns on location data

Our results contain various insights. Some of these insights were used to highlight why our solution of interoperable positioning systems is needed. Figure E.1 shows the user concerns for location data. Based on these results, we can see that the selling of location data to third parties is one of the main concerns. Interoperable positioning systems would provide more transparency towards the collection of this data, and would ultimately open up future work for optimising the privacy of this collected data.

Bibliography

[1] Scott Jenson. Mobile Apps Must Die, 2011. URL https://jenson.org/mobile-apps-must-die/. Accessed on November 2023.

[2] Yanying Gu, Anthony Lo, and Ignas Niemegeers. A Survey of Indoor Positioning Systems for Wireless Personal Networks. *IEEE Communications Surveys & Tutorials*, 11(1), 2009. doi: 10.1109/SURV.2009.090103.

[3] Knut Blind. The impact of standardisation and standards on innovation. In Jakob Edler, Paul Cunningham, Abdullah Gök, and Philip Shapira, editors, *Handbook of Innovation Policy Impact*, Chapters, chapter 14, pages 423–449. Edward Elgar Publishing, 2016. doi: 10.4337/9781784711856.00021.

[4] Maxim Van de Wynckel and Beat Signer. Survey on the Privacy and Transparency of Location Data, May 2025. doi: 10.5281/zenodo.15564050.

[5] Dan Cvrcek, Marek Kumpost, Vashek Matyas, and George Danezis. A Study on the Value of Location Privacy. In *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society (WPES)*, page 109–118. ACM New York, NY, USA, 2006. doi: 10.1145/1179601.1179621.

[6] George Danezis, Stephen Lewis, and Ross J Anderson. How Much is Location Privacy Worth? In *Proceedings of the Workshop on the Economics of Information Security (WEIS)*, volume 5, page 56, 2005.

[7] Google LLC. How Google Uses Location Information – Privacy & Terms – Google, May 2024. URL https://policies.google.com/technologies/location-data?hl=en-US. Accessed on May 2024.

[8] Apple Inc. Legal - Location Services & Privacy - Apple, May 2024. URL https://www.apple.com/legal/privacy/data/en/location-services/. Accessed on May 2024.

[9] Council of the European Union and European Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation). *Official Journal of the European Union*, 59 (119), May 2016. URL https://op.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1.

[10] Publications Office of the European Union. Regulation (EU) 2022/868 of the European Parliament and of the Council of 30 May 2022 on European Data Governance and Amending Regulation (EU) 2018/1724 (Data Governance Act). *Official Journal of the European Union*, pages 1–45, June 2022. ISSN 1977-0677.

[11] Ankit Guptal. Indoor Positioning and Navigation System Market, by Component, by Technology by Platform, Forecast to 2032. Technical Report MRFR/ICT/1243-CR, Market Research Future, October 2018. URL https://www.marketresearchfuture.com/reports/indoor-positioning-navigation-system-market-1775.

[12] Grand View Research Inc. Indoor Positioning And Navigation Market Size, Share, & Trends Analysis Report By Component, By Technology, By Application, By End-use, By Region, And Segment Forecasts, 2024 - 2030. Technical Report GVR-4-68040-384-4, 2024. URL https://www.grandviewresearch.com/industry-analysis/indoor-positioning-navigation-market-report.

[13] MarketsandMarkets Research Pvt. Ltd. Indoor Location Market Size, Share, Growth Analysis, By Offering, Technology, Application, Vertical and Region, Global Forecast to 2029. Technical Report TC 2878, August 2024. URL https://www.marketsandmarkets.com/Market-Reports/indoor-location-market-989.html.

[14] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Peter A.C Heringa, Jaap and't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3(1):160018, March 2016. doi: 10.1038/sdata.2016.18.

[15] Ivana Ivánová, Ryan Keenan, Christopher Marshall, Lori Mancell, Eldar Rubinov, Ryan Ruddick, Nicholas Brown, and Graeme Kernich. FAIR Data and Metadata: GNSS Precise Positioning User Perspective. *Data Intelligence*, 5(1):43–74, March 2023. doi: 10.1162/dint_a_00185.

[16] Publications Office of the European Union. European Digital Markets Act. *Official Journal of the European Union*, June 2023. URL https://op.europa.eu/en/publication-detail/-/publication/d1c08e1b-c74d-11ed-a05c-01aa75ed71a1.

[17] Sandra Heiler. Semantic Interoperability. *ACM Computing Surveys (CSUR)*, 27(2): 271–273, 1995. doi: 10.1145/210376.210392.

[18] Peter Wegner. Interoperability. *ACM Computing Surveys (CSUR)*, 28(1):285–287, 1996. doi: 10.1145/234313.234424.

[19] ISO Central Secretary. Intelligent Transport Systems — Indoor Navigation for Personal and Vehicle ITS Station – Part 2: Requirements and Specification for Indoor Maps. Standard ISO 17438-2:2024, International Organization for Standardization, September 2024. URL https://www.iso.org/standard/67087.html.

[20] ISO Central Secretary. Intelligent Transport Systems — Indoor Navigation for Personal and Vehicle ITS Station – Part 3: Requirements and Specification for Indoor Positioning Reference Data. Standard ISO 17438-3:2024, International Organization for Standardization, September 2024. URL https://www.iso.org/standard/67088.html.

[21] ISO Central Secretary. Intelligent Transport Systems — Indoor Navigation for Personal and Vehicle ITS Station – Part 4: Requirements and Specifications for Interface Between Personal/Vehicle and Central ITS Stations. Standard ISO 17438-4:2019, International Organization for Standardization, 2019. URL https://www.iso.org/standard/67089.html.

[22] ISO Central Secretary. Information technology – Real time locating systems — Test and evaluation of localization and tracking systems. Standard ISO 18305:2016, International Organization for Standardization, 2016. URL https://www.iso.org/standard/62090.html.

[23] Pooyan Shams Farahsari, Amirhossein Farahzadi, Javad Rezazadeh, and Alireza Bagheri. A Survey on Indoor Positioning Systems for IoT-Based Applications. *IEEE Internet of Things Journal*, 9(10):7680–7699, 2022. doi: 10.1109/JIOT.2022.3149048.

[24] Yuejin Deng, Haojun Ai, Zeyu Deng, Wenxiu Gao, and Jianga Shang. An Overview of Indoor Positioning and Mapping Technology Standards. *Standards*, 2(2):157–183, 2022. ISSN 2305-6703. doi: 10.3390/standards2020012.

[25] Francesco Furfari, Antonino Crivello, Paolo Barsocchi, Filippo Palumbo, and Francesco Potortì. What is next for Indoor Localisation? Taxonomy, protocols, and patterns for advanced Location Based Services. In *Proceedings of the 10th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2019)*, pages 1–8, 2019. doi: 10.1109/IPIN.2019.8911759.

[26] Francesco Furfari, Antonino Crivello, Paolo Baronti, Paolo Barsocchi, Michele Girolami, Filippo Palumbo, Darwin Quezada-Gaibor, Germán M. Mendoza Silva, and Joaquín Torres-Sospedra. Discovering Location Based Services: A Unified Approach for Heterogeneous Indoor Localization Systems. *Internet of Things*, 13: 100334, 2021. doi: 10.1016/j.iot.2020.100334.

[27] Mohab Aly, Foutse Khomh, Yann-Gaël Guéhéneuc, Hironori Washizaki, and Soumaya Yacout. Is Fragmentation a Threat to the Success of the Internet of Things? *IEEE Internet of Things Journal*, 6(1):472–487, 2019. doi: 10.1109/JIOT.2018.2863180.

[28] Michael Strasser and Sahin Albayrak. Conceptual Architecture for Self-discovering in Fragmented Service Systems. In *Proceedings of the 7th International Conference on New Technologies, Mobility and Security (NTMS 2015)*, pages 1–5, 2015. doi: 10.1109/NTMS.2015.7266479.

[29] Xiansheng Guo, Nirwan Ansari, Lin Li, and Linfu Duan. A Hybrid Positioning System for Location-Based Services: Design and Implementation. *IEEE Communications Magazine*, 58(5):90–96, 2020. doi: 10.1109/MCOM.001.1900737.

[30] Kim H. Veltman. Syntactic and Semantic Interoperability: New Approaches to Knowledge and the Semantic Web. *New Review of Information Networking*, 7(1): 159–183, 2001. doi: 10.1080/13614570109516975.

[31] Altti Ilari Maarala, Xiang Su, and Jukka Riekki. Semantic Reasoning for Context-Aware Internet of Things Applications. *IEEE Internet of Things Journal*, 4(2): 461–473, 2017. doi: 10.1109/JIOT.2016.2587060.

[32] Maxim Van de Wynckel and Beat Signer. OpenHPS: An Open Source Hybrid Positioning System. Technical Report WISE-2020-01, Vrije Universiteit Brussel, 2020. doi: 10.48550/ARXIV.2101.05198.

[33] Maxim Van de Wynckel and Beat Signer. Indoor Positioning Using the OpenHPS Framework. In *Proceedings of the 11th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2021)*, 2021. doi: 10.1109/IPIN51156.2021.9662569.

[34] Maxim Van de Wynckel and Beat Signer. OpenHPS: A Modular Framework to Facilitate the Development of FAIR Positioning Systems. *Journal of Open Source Software*, 10(110):8113, June 2025. doi: 10.21105/joss.08113.

[35] Maxim Van de Wynckel and Beat Signer. OpenHPS: Single Floor Fingerprinting and Trajectory Dataset, May 2021. doi: 10.5281/zenodo.4744380.

[36] Benjamin Vermunicht, Maxim Van de Wynckel, and Beat Signer. 802.11 Managemement Frames From a Public Location, June 2023. doi: 10.5281/zenodo.8003771.

[37] Nathan Hoebeke, Maxim Van de Wynckel, and Beat Signer. Object Tracking on a Monopoly Game Board, June 2023. doi: 10.5281/zenodo.7990434.

[38] Maxim Van de Wynckel. Garage Positioning Dataset, 2025. doi: 10.34740/KAGGLE/DS/6654647.

[39] Maxim Van de Wynckel and Beat Signer. Sphero Dead Reckoning and CV Tracking Dataset, 2025. doi: 10.34740/KAGGLE/DS/6760212.

[40] Maxim Van de Wynckel and Beat Signer. POSO: A Generic Positioning System Ontology. In *Proceedings of the 21st International Semantic Web Conference (ISWC 2022)*, Virtual conference, 2022. doi: 10.1007/978-3-031-19433-7_14.

[41] Maxim Van de Wynckel, Isaac Valadez, and Beat Signer. FidMark: A Fiducial Marker Ontology for Semantically Describing Visual Markers. In *Proceedings of The Semantic Web (ESWC 2024)*, 2024. doi: 10.1007/978-3-031-60635-9_14.

[42] Maxim Van de Wynckel and Beat Signer. Discoverable and Interoperable Augmented Reality Environments Through Solid Pods. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL https://ceur-ws.org/Vol-3947/short2.pdf.

[43] Maxim Van de Wynckel and Beat Signer. A Solid-based Architecture for Decentralised Interoperable Location Data. In *Proceedings of the 12th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2022), CEUR Workshop Proceedings*, volume 3248, 2022. URL https://ceur-ws.org/Vol-3248/paper11.pdf.

[44] Yoshi Malaise, Maxim Van de Wynckel, and Beat Signer. Towards Distributed Intelligent Tutoring Systems Based on User-owned Progress and Performance Data. In *Proceedings of the 2nd Solid Symposium (SoSy 2024)*, volume 3947, May 2024. URL https://ceur-ws.org/Vol-3947/short3.pdf.

[45] Maxim Van de Wynckel and Beat Signer. SemBeacon: A Semantic Proximity Beacon Solution for Discovering and Detecting the Position of Physical Things. In *Proceedings of the 13th International Conference on the Internet of Things (IoT 2023)*, 2023. doi: 10.1145/3627050.3627060.

[46] ISO Central Secretary. Geographic information – Positioning Services. Standard ISO 19116:2019, International Organization for Standardization, 2019. URL https://www.iso.org/standard/70882.html.

[47] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and Elmar Wasle. *GNSS–Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media, 2007. ISBN 9783211730171.

[48] Goran M. Djuknic and Robert E. Richton. Geolocation and Assisted GPS. *Computer*, 34(2), 2001. doi: 10.1109/2.901174.

[49] John Krumm, Steve Harris, Brian Meyers, Barry Brumitt, Michael Hale, and Steve Shafer. Multi-Camera Multi-Person Tracking for EasyLiving. In *Proceedings of the 3rd IEEE International Workshop on Visual Surveillance*, pages 3–10, 2000. doi: 10.1109/VS.2000.856852.

[50] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A Versatile Visual SLAM Framework. In *Proceedings of MM 2019, International Conference on Multimedia*, Nice, France, October 2019. doi: 10.1145/3343031.3350539.

[51] Christina Gsaxner, Jianning Li, Antonio Pepe, Dieter Schmalstieg, and Jan Egger. Inside-out Instrument Tracking for Surgical Navigation in Augmented Reality. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology (VRST 2021)*, pages 1–11. ACM New York, NY, USA, 2021. doi: 10.1145/3489849.3489863.

[52] Kai-Tai Song and Yueh Chuan Chang. Design and Implementation of a Pose Estimation System Based on Visual Fiducial Features and Multiple Cameras. In *Proceedings of the 2018 International Automatic Control Conference (CACS 2018)*, pages 1–6. IEEE, 2018. doi: 10.1109/CACS.2018.8606773.

[53] Robert Harle. A Survey of Indoor Inertial Positioning Systems for Pedestrians. *IEEE Communications Surveys & Tutorials*, 15(3):1281–1293, 2013. doi: 10.1109/SURV.2012.121912.00075.

[54] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007. doi: 10.1109/TSMCC.2007.905750.

[55] Piotr Mirowski, Tin Kam Ho, Saehoon Yi, and Michael MacDonald. SignalSLAM: Simultaneous Localization and Mapping with Mixed WiFi, Bluetooth, LTE and Magnetic Signals. In *Proceedings of the 4th International Conference on Indoor Positioning and Indoor Navigation*, Montbeliard-Belfort, France, October 2013. doi: 10.1109/IPIN.2013.6817853.

[56] Zhenghua Chen, Han Zou, Hao Jiang, Qingchang Zhu, Yeng Soh, and Lihua Xie. Fusion of WiFi, Smartphone Sensors and Landmarks Using the Kalman Filter for Indoor Localization. *Sensors*, 15(1), January 2015. doi: 10.3390/s150100715.

[57] Anja Bekkelien and Michel Deriaz. Hybrid Positioning Framework for Mobile Devices. In *Proceedings of UPINLBS 2012, International Conference on Ubiquitous Positioning, Indoor Navigation, and Location Based Service*, Helsinki, Finland, October 2012. doi: 10.1109/UPINLBS.2012.6409759.

[58] Massimo Ficco and Stefano Russo. A Hybrid Positioning System for Technology-independent Location-aware Computing. *Software: Practice and Experience*, 39 (13), September 2009. doi: 10.1002/spe.919.

[59] Sean J. Barbeau, Miguel A. Labrador, Philip L. Winters, Rafael Pérez, and Nevine Labib Georggi. Location API 2.0 for J2ME: A New Standard in Location for Java-enabled Mobile Phones. *Computer Communications*, 31(6), April 2008. doi: 10.1016/j.comcom.2008.01.045.

[60] Philipp M. Scholl, Stefan Kohlbrecher, Vinay Sachidananda, and Kristof Van Laerhoven. Fast Indoor Radio-Map Building for RSSI-based Localization Systems. In *Proceedings of the International Conference on Networked Sensing (INSS 2012)*, Antwerp, Belgium, June 2012. doi: 10.1109/INSS.2012.6240574.

[61] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An Open-Source Robot Operating System. In *Proceedings of the International Workshop on Open Source Software*, Kobe, Japan, May 2009.

[62] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time Human Pose Recognition in Parts From Single Depth Images. *Commun. ACM*, 56(1):116–124, January 2013. doi: 10.1145/2398356.2398381.

[63] Jochen Schiller and Agnès Voisard. *Location-based Services*. Elsevier, 2004. doi: 10.1016/B978-1-55860-929-7.X5000-6.

[64] Andrei Popescu. Geolocation API Specification 2nd Edition. Specification, World Wide Web Consortium (W3C), November 2016. URL https://www.w3.org/TR/geolocation-API/.

[65] B. Louis Decker. World Geodetic System 1984. In *Proceedings to the Fourth International Geodetic Symposium on Satellite Positioning*, Austin, USA, April 1986.

[66] Cristiano di Flora, Massimo Ficco, Stefano Russo, and Vincenzo Vecchio. Indoor and Outdoor Location Based Services for Portable Wireless Devices. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (SDCS 2005)*, Columbus, USA, June 2005. doi: 10.1109/ICDCSW.2005.77.

[67] Joel Luis Carbonera, Sandro Rama Fiorini, Edson Prestes, Vitor AM Jorge, Mara Abel, Raj Madhavan, Angela Locoro, Paulo Gonçalves, Tamás Haidegger, Marcos E Barreto, et al. Defining Positioning in a Core Ontology for Robotics. In *Proceedings of IROS 2013, IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1867–1872, 2013. doi: 10.1109/IROS.2013.6696603.

[68] Madhawa Perera, Armin Haller, Sergio José Rodríguez Méndez, and Matt Adcock. HDGI: A Human Device Gesture Interaction Ontology for the Internet of Things. In *Proceedings of the 19th International Semantic Web Conference (ISWC 2020)*, pages 111–126, 2020. doi: 10.1007/978-3-030-62466-8_8.

[69] Hashim A. Hashim. Special Orthogonal Group SO(3), Euler Angles, Angle-axis, Rodriguez Vector and Unit-Quaternion: Overview, Mapping and Challenges, 2021.

[70] ISO Central Secretary. Information Technology – Database Languages – SQL Multimedia and Application Packages – Part 3: Spatial. Standard ISO/IEC 13249-3:2016, International Organization for Standardization, Geneva, Switzerland, 2016. URL https://www.iso.org/standard/60343.html.

[71] James Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. *Matrix*, 58(15-16):1–35, 2006. URL https://api.semanticscholar.org/CorpusID:16450526.

[72] Fabian Hölzke, Johann-P. Wolff, and Christian Haubelt. Improving Pedestrian Dead Reckoning Using Likely Paths and Backtracking for Mobile Devices. In *Proceedings of the International Workshop on Pervasive Smart Living Spaces (PerLS 2019)*, Kyoto, Japan, March 2019. doi: 10.1109/PERCOMW.2019.8730734.

[73] Álvaro De-La-Llana-Calvo, José-Luis Lázaro-Galilea, Alfredo Gardel-Vicente, David Rodríguez-Navarro, Ignacio Bravo-Muñoz, and Felipe Espinosa-Zapata. Characterization of Multipath Effects in Indoor Positioning Systems by AoA and PoA Based on Optical Signals. *Sensors*, 19(4), 2019. doi: 10.3390/s19040917.

[74] Wilson Sakpere, Michael Adeyeye-Oshin, and Nhlanhla BW Mlitwa. A State-of-the-art Survey of Indoor Positioning and Navigation Systems and Technologies. *South African Computer Journal*, 29(3):145–197, 2017.

[75] Janne Haverinen. Utilizing Magnetic Field Based Navigation, August 2014. US Patent 8,798,924.

[76] Seong-Eun Kim, Yong Kim, Jihyun Yoon, and Eung Sun Kim. Indoor Positioning System Using Geomagnetic Anomalies for Smartphones. In *Proceedings of the 2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN 2012)*, pages 1–5, 2012. doi: 10.1109/IPIN.2012.6418947.

[77] Sudeep Pasricha, Viney Ugave, Charles W Anderson, and Qi Han. LearnLoc: A Framework for Smart Indoor Localization with Embedded Mobile Devices. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES 2015)*, Amsterdam, Netherlands, October 2015. doi: 10.1109/CODESISSS.2015.7331366.

[78] Marvelmind Robotics. Typical Costs of Indoor Positioning System, March 2024. URL https://marvelmind.com/download/costs/.

[79] Kris Luyten and Karin Coninx. ImogI: Take Control Over a Context-aware Electronic Mobile Guide for Museums. In *Proceedings of the Workshop on HCI in Mobile Guides, in conjunction with 6th International Conference on Human Computer Interaction with Mobile Devices and Services*. Citeseer, 2004.

[80] Ching-Sheng Wang, Ding-Jung Chiang, and Yi-Yun Ho. 3D Augmented Reality Mobile Navigation System Supporting Indoor Positioning Function. In *Proceedings of the 2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom)*, pages 64–68, 2012. doi: 10.1109/CyberneticsCom.2012.6381618.

[81] Marius Huguet, Canan Pehlivan, François Ballereau, Antoine Dodane-Loyenet, Franck Fontanili, Thierry Garaix, Youri Yordanov, Vincent Augusto, Karim

Tazarourte, and Abdesslam Redjaline. Indoor Positioning Systems Provide Insight Into Emergency Department Systems Enabling Proposal of Designs to Improve Workflow. *Communications Medicine*, 5(1):72, Mar 2025. ISSN 2730-664X. doi: 10.1038/s43856-025-00793-y. URL https://doi.org/10.1038/s43856-025-00793-y.

[82] Maged N. Kamel Boulos and Geoff Berry. Real-time Locating Systems (RTLS) in Healthcare: A Condensed Primer. *International Journal of Health Geographics*, 11 (1):25, Jun 2012. doi: 10.1186/1476-072X-11-25.

[83] Laura Vaccari, Antonio Maria Coruzzolo, Francesco Lolli, and Miguel Afonso Sellitto. Indoor Positioning Systems in Logistics: A Review. *Logistics*, 8(4), 2024. doi: 10.3390/logistics8040126.

[84] Hyunwoo Hwangbo, Jonghyuk Kim, Zoonky Lee, and Soyean Kim. Store Layout Optimization Using Indoor Positioning System. *International Journal of Distributed Sensor Networks*, 13(2), 2017. doi: 10.1177/1550147717692585.

[85] Sony. Nimway from Sony - Your Smart Office Solution, 2025. URL https://www.sonynetworkcom.com/nimway. Accessed on February 2025.

[86] Ling Pei, Ruizhi Chen, Yuewei Chen, Helena Leppäkoski, and Arto Perttula. Indoor/Outdoor Seamless Positioning Technologies Integrated on Smart Phone. In *Proceedings of the First International Conference on Advances in Satellite and Space Communications (SPACOMM 2019)*, pages 141–145. IEEE, 2009. doi: 10.1109/SPACOMM.2009.12.

[87] Naohiko Kohtake, Shusuke Morimoto, Satoshi Kogure, and Dinesh Manandhar. Indoor and Outdoor Seamless Positioning Using Indoor Messaging System and GPS. In *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2011), Guimarães, Portugal*, pages 21–23, 2011.

[88] Danijel Čabarkapa, Ivana Grujić, and Petar Pavlović. Comparative Analysis of the Bluetooth Low-Energy Indoor Positioning Systems. In *Proceedings of the 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS 2015)*, 2015. doi: 10.1109/TELSKS.2015.7357741.

[89] Cole Gleason et al. Crowdsourcing the Installation and Maintenance of Indoor Localization Infrastructure to Support Blind Navigation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1), 2018. doi: 10.1145/3191741.

[90] Kang Eun Jeon et al. BLE Beacons for Internet of Things Applications: Survey, Challenges, and Opportunities. *IEEE Internet of Things Journal*, 5(2), 2018. doi: 10.1109/JIOT.2017.2788449.

[91] Bluetooth SIG Inc. Bluetooth Core Specification v5.1. Specification, Bluetooth SIG Inc., 2020. URL https://bluetooth.com/specifications/specs/core-specification-5-1/.

[92] Maeve Kennedy, Petros Spachos, and Graham W. Taylor. BLE Beacon Indoor Localization Dataset, 2019. doi: 10.5683/SP2/UTZTFT.

[93] Chun Yang and Andrey Soloviev. Positioning with Mixed Signals of Opportunity Subject to Multipath and Clock Errors in Urban Mobile Fading Environments. In

Proceedings of the 31st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2018), pages 223–243, 2018. doi: 10.33012/2018.15833.

[94] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom 2009)*, page 261–272. ACM New York, NY, USA, 2009. doi: 10.1145/1614320.1614350.

[95] Wendong Xiao, Wei Ni, and Yue Khing Toh. Integrated Wi-Fi Fingerprinting and Inertial Sensing for Indoor Positioning. In *Proceedings of the 2nd International Conference on Indoor Positioning and Indoor Navigation (IPIN 2011)*, Guimaraes, Portugal, September 2011. doi: 10.1109/IPIN.2011.6071921.

[96] Xuesheng Peng, Ruizhi Chen, Kegen Yu, Feng Ye, and Weixing Xue. An Improved Weighted K-nearest Neighbor Algorithm for Indoor Localization. *Electronics*, 9 (12):2117, 2020. doi: 10.3390/electronics9122117.

[97] Feng Qin, Tao Zuo, and Xing Wang. CCpos: WiFi Fingerprint Indoor Positioning System Based on CDAE-CNN. *Sensors*, 21(4):1114, 2021. doi: 10.3390/s21041114.

[98] Alwin Poulose, Jihun Kim, and Dong Seog Han. Indoor Localization with Smartphones: Magnetometer Calibration. In *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE 2019)*, pages 1–3, 2019. doi: 10.1109/ICCE.2019.8661986.

[99] Maxim Van de Wynckel. Indoor Navigation by Centralized Tracking. Master's thesis, Vrije Universiteit Brussel, Elsene, Belgium, June 2019. Available at https://researchportal.vub.be/en/studentTheses/indoor-navigation-by-centralized-tracking.

[100] Stephane Beauregard and Harald Haas. Pedestrian Dead Reckoning: A Basis for Personal Positioning. In *Proceedings of the 3rd Workshop on Positioning, Navigation and Communication (WPNC 2006)*, Merida City, Mexico, October 2006. doi: 10.1109/ICEEE.2011.6106608.

[101] J.A Cooney, W.L Xu, and G Bright. Visual Dead-reckoning for Motion Control of a Mecanum-wheeled Mobile Robot. *Mechatronics*, 14(6):623–637, 2004. doi: 10.1016/j.mechatronics.2003.09.002.

[102] Wilfried Elmenreich. An Introduction to Sensor Fusion. Technical Report 47/2001, Vienna University of Technology, November 2002.

[103] Walter T Higgins. A Comparison of Complementary and Kalman Filtering. *IEEE Transactions on Aerospace and Electronic Systems*, AES-11(3):321–325, 1975. doi: 10.1109/TAES.1975.308081.

[104] Xudong Song, Xiaochen Fan, Xiangjian He, Chaocan Xiang, Qianwen Ye, Xiang Huang, Gengfa Fang, Liming Luke Chen, Jing Qin, and Zumin Wang. CNNLoc: Deep-Learning Based Indoor Localization with WiFi Fingerprinting. In *Proceedings of the 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innova-*

tion (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pages 589–595. IEEE, 2019. doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00139.

[105] Martin Brossard, Axel Barrau, and Silvère Bonnabel. AI-IMU Dead-Reckoning. *IEEE Transactions on Intelligent Vehicles*, 5(4):585–595, 2020. doi: 10.1109/TIV.2020.2980758.

[106] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep Kalman Filters. *arXiv preprint arXiv:1511.05121*, 2015. doi: 10.48550/arXiv.1511.05121.

[107] Nancy Ide and James Pustejovsky. What Does Interoperability Mean, Anyway? Toward an Operational Definition of Interoperability for Language Technology. In *Proceedings of the Second International Conference on Global Interoperability for Language Resources. Hong Kong, China*, 2010. URL https://api.semanticscholar.org/CorpusID:11018727.

[108] Wajahat Ali Khan, Maqbool Hussain, Khalid Latif, Muhammad Afzal, Farooq Ahmad, and Sungyoung Lee. Process Interoperability in Healthcare Systems With Dynamic Semantic Web Services. *Computing*, 95:837–862, 2013. doi: 10.1007/s00607-012-0239-3.

[109] Elivaldo Lozer Fracalossi Ribeiro, Erasmo Leite Monteiro, Daniela Barreiro Claro, and Rita Suzana Pitangueira Maciel. A Conceptual Framework for Pragmatic Interoperability. In *Proceedings of the 15th Brazilian Symposium on Information Systems (SBSI 2019)*, SBSI '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450372374. doi: 10.1145/3330204.3330246.

[110] Alek Radjenovic and Richard F. Paige. Behavioural Interoperability to Support Model-Driven Systems Integration. In *Proceedings of the First International Workshop on Model-Driven Interoperability (MDI 2010)*, MDI '10, page 98–107, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302920. doi: 10.1145/1866272.1866285.

[111] Jochen Schiller and Agnès Voisard. LBS Interoperability Through Standards. In *Location-based Services*, pages 149–172. Elsevier, 2004. doi: 10.1016/B978-1-55860-929-7.X5000-6.

[112] Marwa Mabrouk, Tom Bychowski, Jonathan Williams, Harry Niedzwiadek, Yaser Bishr, Jean-Francois Gaillet, Neil Crisp, Will Wilbrink, Mike Horhammer, Greg Roy, Serge Margoulies, Gil Fuchs, and Geoffrey Hendrey. OpenGIS Location Services (OpenLS): Core Services. Technical Report OGC 07-074, Open Geospatial Consortium, September 2008. URL https://portal.ogc.org/files/?artifact_id=22122.

[113] Jong-Woo Kim, Ju-Yeon Kim, Hyun-Suk Hwang, Sung-Seok Park, Chang-Soo Kim, and Sung-gi Park. The Semantic Web Approach in Location Based Services. In *Proceedings of Computational Science and Its Applications (ICCSA 2005)*, pages 127–136. Springer Berlin Heidelberg, 2005. doi: 10.1007/11424826_14.

[114] Guowei Cai, Ben M. Chen, and Tong Heng Lee. *Coordinate Systems and Transformations*, pages 23–34. Springer London, London, 2011. doi: 10.1007/978-0-85729-635-1_2.

[115] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. GeoJSON, 2014. URL https://geojson.org. Accessed on January 2020.

[116] ISO Central Secretary. Geographic Information—Well-known Text Representation of Coordinate Reference Systems. Standard ISO 19162:2019, International Organization for Standardization, Geneva, Switzerland, July 2019. URL https://www.iso.org/standard/76496.html.

[117] ISO Central Secretary. Geographic information —- Geography Markup Language (GML). Standard ISO 19136-1:2019, International Organization for Standardization, 2020. URL https://www.iso.org/standard/75676.html.

[118] Roberto Minerva, Gyu Myoung Lee, and Noel Crespi. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of IEEE*, 108(10):1785–1824, 2020. doi: 10.1109/JPROC.2020.2998530.

[119] Hubert Lehner and Lionel Dorffner. Digital geoTwin Vienna: Towards a Digital Twin City as Geodata Hub. *Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1):63–75, February 2020. doi: 10.1007/s41064-020-00101-4.

[120] OGC Community Standard Apple Inc. Indoor Mapping Data Format. Technical report, Open Geospatial Consortium (OGC), 2021. URL https://docs.ogc.org/cs/20-094/index.html.

[121] Hae-Kyong Kang and Ki-Joune Li. A Standard Indoor Spatial Data Model—OGC IndoorGML and Implementation Approaches. *ISPRS International Journal of Geo-Information*, 6(4), 2017. doi: 10.3390/ijgi6040116.

[122] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011. ISBN 9781449319915.

[123] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP: Building Distributed Applications*. "O'Reilly Media, Inc.", 2001. ISBN 978-0596000950.

[124] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[125] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin Core Metadata for Resource Discovery. Technical report, The Internet Society, 1998.

[126] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), September 2009. doi: 10.1145/1567274.1567278.

[127] Olaf Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proceedings of The Semantic Web: Research and Applications (ESWC 2011)*, pages 154–169. Springer Berlin Heidelberg, 2011.

[128] Matthew Perry, John Herring, Nicholas J. Car, Timo Homburg, Simon J.D. Cox, Matthias Bonduel, and Frans Knibbe. OGC GeoSPARQL: A Geographic Query Language for RDF Data: GeoSPARQL 1.1 Draft. Technical report, Open Geospatial Consortium (OGC), 2021. URL https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html.

[129] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption. In *Companion Proceedings of the Web Conference (WWW 2022)*, page 260–263. ACM New York, NY, USA, 2022. doi: 10.1145/3487553.3524253.

[130] Krzysztof Janowicz and Michael Compton. The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration Into the Semantic Sensor Network Ontology. In *Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN 2010)*, volume 668, 2010. URL https://ceur-ws.org/Vol-668/paper12.pdf.

[131] Krzysztof Janowicz, Armin Haller, Simon JD Cox, Danh Le Phuoc, and Maxime Lefrançois. SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators. *Journal of Web Semantics*, 56, May 2019. doi: 10.1016/j.websem.2018.06.003.

[132] Anna Nguyen, Tobias Weller, Michael Färber, and York Sure-Vetter. Making Neural Networks FAIR. In *Proceedings of Knowledge Graphs and Semantic Web: Second Iberoamerican Conference and First Indo-American Conference (KGSWC 2020)*, pages 29–44. Springer, 2020. doi: 10.1007/978-3-030-65384-2_3.

[133] Kassem Fawaz and Kang G Shin. Location Privacy Protection for Smartphone Users. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014)*, pages 239–250, 2014. doi: 10.1145/2660267.2660270.

[134] Hongbo Jiang, Jie Li, Ping Zhao, Fanzi Zeng, Zhu Xiao, and Arun Iyengar. Location Privacy-Preserving Mechanisms in Location-based Services: A Comprehensive Survey. *ACM Computing Surveys (CSUR)*, 54(1):1–36, 2021. doi: 10.1145/3423165.

[135] Sofie Verbrugge, Frederic Vannieuwenborg, Marlies Van der Wee, Didier Colle, Ruben Taelman, and Ruben Verborgh. Towards a Personal Data Vault Society: An Interplay Between Technological and Business Perspectives. In *Proceedings of the 60th FITCE Communication Days Congress for ICT Professionals: Industrial Data–Cloud, Low Latency and Privacy (FITCE)*, pages 1–6. IEEE, 2021. doi: 10.1109/FITCE53297.2021.9588540.

[136] Sofie Verbrugge, Frederic Vannieuwenborg, Marlies Van der Wee, Didier Colle, Ruben Taelman, and Ruben Verborgh. Towards a Personal Data Vault Society: An Interplay Between Technological and Business Perspectives. In *Proceedings of the 60th FITCE Communication Days Congress for ICT Professionals: Industrial Data – Cloud, Low Latency and Privacy (FITCE 2021)*, pages 1–6, 2021. doi: 10.1109/FITCE53297.2021.9588540.

[137] Ravi S Sandhu and Pierangela Samarati. Access Control: Principle and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1994. doi: 10.1109/35.312842.

[138] Min Y Mun, Donnie H Kim, Katie Shilton, Deborah Estrin, Mark Hansen, and Ramesh Govindan. PDVLoc: A Personal Data Vault for Controlled Location Data Sharing. *ACM Transactions on Sensor Networks (TOSN)*, 10(4):1–29, 2014. doi: 10.1145/2523820.

[139] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. Solid: A Platform for Decentralized Social Applications Based on Linked Data. Technical report, MIT CSAIL & QCRI, 2016.

[140] Andrei Sambra, Henry Story, and Tim Berners-Lee. WebID 1.0. W3C Editor's Draft, World Wide Web Consortium (W3C), 2014. URL https://www.w3.org/2005/Incubator/webid/spec/identity/.

[141] Eugene Ferry, John O Raw, and Kevin Curran. Security Evaluation of the OAuth 2.0 Framework. *Information & Computer Security*, 23(1):73–101, March 2015. doi: 10.1108/ics-12-2013-0089.

[142] Zirui Hu, Yuhan Yang, Jing Wu, and Chengnian Long. A Secure and Efficient Blockchain-Based Data Sharing Scheme for Location Data. In *Proceedings of the 2022 4th International Conference on Blockchain Technology (ICBT 2022)*, page 110–116. ACM New York, NY, USA, 2022. doi: 10.1145/3532640.3532655.

[143] Muhammad Imran Sarwar, Muhammad Waseem Iqbal, Tahir Alyas, Abdallah Namoun, Ahmed Alrehaili, Ali Tufail, and Nadia Tabassum. Data Vaults for Blockchain-Empowered Accounting Information Systems. *IEEE Access*, 9:117306–117324, 2021. doi: 10.1109/ACCESS.2021.3107484.

[144] Salam Traboulsi. Overview of 5G-oriented Positioning Technology in Smart Cities. *Procedia Computer Science*, 201:368–374, 2022. ISSN 1877-0509. doi: 10.1016/j.procs.2022.03.049. Proceedings of the 13th International Conference on Ambient Systems, Networks and Technologies (ANT 2022) and the 5th International Conference on Emerging Data and Industry 4.0 (EDI40).

[145] Salil Pradhan. Semantic Location. *Personal Technologies*, 4(4), 2000. doi: 10.1007/BF02391560.

[146] Tim Kindberg and John Barton. A Web-based Nomadic Computing System. *Computer Networks*, 35(4), 2001. doi: 10.1016/S1389-1286(00)00181-X.

[147] Vassilis Papataxiarhis et al. MNISIKLIS: Indoor Location Based Services for All. *Location Based Services and TelecCrtography II*, 2008. doi: 10.1007/978-3-540-87393-8_16.

[148] Sanya Khruahong et al. Multi-Level Indoor Navigation Ontology for High Assurance Location-based Services. In *Proceedings of HASE 2017*, 2017. doi: 10.1109/HASE.2017.9.

[149] Kangjae Lee, Jiyeong Lee, and Mei-Po Kwan. Location-based Service Using Ontology-based Semantic Queries: A Study With a Focus on Indoor Activities in a University Context. *Computers, Environment and Urban Systems*, 62, 2017. doi: 10.1016/j.compenvurbsys.2016.10.009.

[150] Sujith Samuel Mathew et al. Web of Things: Description, Discovery and Integration. In *Proceedings of IoT'11 and CPSCom'11*, 2011. doi: 10.1109/iThings/CPSCom.2011.165.

[151] Dave Raggett. The Web of Things: Challenges and Opportunities. *Computer*, 48 (5), 2015. doi: 10.1109/MC.2015.149.

[152] Dmitry Namiot and Manfred Sneps-Sneppe. The Physical Web in Smart Cities. In *Proceedings of the 2015 Advances in Wireless and Optical Communications (RTUWO 2015)*, 2015. doi: 10.1109/RTUWO.2015.7365717.

[153] Google LLC. UriBeacon: The Web Uri Open Beacon Specification for the Internet of Things, 2015. URL https://github.com/google/uribeacon/tree/uribeacon-final. Accessed on November 2022.

[154] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: A Data Discovery System. In *Proceedings of the 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012, 2018. doi: 10.1109/ICDE.2018.00094.

[155] Dmitry Namiot and Manfred Sneps-Sneppe. Context-aware data discovery. In *Proceedings of the 16th International Conference on Intelligence in Next Generation Networks*, pages 134–141, 2012. doi: 10.1109/ICIN.2012.6376016.

[156] K. Arabshian and H. Schulzrinne. GloServ: Global Service Discovery Architecture. In *Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS 2004)*, pages 319–325, 2004. doi: 10.1109/MOBIQ.2004.1331738.

[157] Christian Bettstetter and Christoph Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In *Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications (EUNICE 2002)*, 2002. URL https://api.semanticscholar.org/CorpusID:13179804.

[158] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. Peer-to-Peer-Based Resource Discovery in Global Grids: A Tutorial. *IEEE Communications Surveys & Tutorials*, 10(2):6–33, 2008. doi: 10.1109/COMST.2008.4564477.

[159] Md Rakib Shahriar, Xiaoqing Frank Liu, Md Mahfuzer Rahman, and SM Nahian Al Sunny. OpenDT: A Reference Framework for Service Publication and Discovery using Remote Programmable Digital Twins. In *Proceedings of the 2020 IEEE International Conference on Services Computing (SCC 2020)*, pages 116–123. IEEE, 2020. doi: 10.1109/SCC49832.2020.00024.

[160] Kyriakos Georgiou, Timotheos Constambeys, Christos Laoudias, Lambros Petrou, Georgios Chatzimilioudis, and Demetrios Zeinalipour-Yazti. Anyplace: A Crowdsourced Indoor Information Service. In *Proceedings of the International Conference on Mobile Data Management: Systems, Services and Middleware (MDM 2015)*, pages 291–294, 2015. doi: 10.1109/MDM.2015.80.

[161] Allal Tiberkak, Abdelfetah Hentout, Mustapha Aouache, and Abdelkader Belkhir. Outdoor UPnP for Services Discovery in Smart Cities. In *Proceedings of the 2018 International Conference on Applied Smart Systems (ICASS 2018)*, pages 1–6. IEEE, 2018. doi: 10.1109/ICASS.2018.8652043.

[162] José M Sánchez Santana, Marina Petrova, and Petri Mahonen. UPnP Service Discovery for Heterogeneous Networks. In *Proceedings of the 17th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2006)*, pages 1–5. IEEE, 2006. doi: 10.1109/PIMRC.2006.254286.

[163] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An Overview on Edge Computing Research. *IEEE Access*, 8:85714–85728, 2020. doi: 10.1109/ACCESS.2020.2991734.

[164] Max J Egenhofer. Toward the Semantic Geospatial Web. In *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, pages 1–4. ACM New York, NY, USA, 2002. doi: 10.1145/585147.585148.

[165] Dongming Guo and Erling Onstein. State-of-the-Art Geospatial Information Processing in NoSQL Databases. *ISPRS International Journal of Geo-Information*,

9(5), 2020. doi: 10.3390/ijgi9050331. URL https://www.mdpi.com/2220-9964/9/5/331.

[166] Antonios Makris, Konstantinos Tserpes, Giannis Spiliopoulos, Dimitrios Zissis, and Dimosthenis Anagnostopoulos. MongoDB VS PostgreSQL: A Comparative Study on Performance Aspects. *GeoInformatica*, 25:243–268, 2021.

[167] Robert Battle and Dave Kolas. GeoSPARQL: Enabling a Geospatial Semantic Web. *Semantic Web Journal*, 3(4), 2011.

[168] Ruchika Muddinagiri, Shubham Ambavane, Vivek Jadhav, and Santosh Tamboli. Proximity Marketing Using Bluetooth Low Energy. In *Proceedings of the 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 856–861, 2020. doi: 10.1109/ICACCS48705.2020.9074160.

[169] L Sofyana and A R Putera. Business Architecture Planning With TOGAF Framework. *Journal of Physics: Conference Series*, 1375(1):012056, November 2019. doi: 10.1088/1742-6596/1375/1/012056.

[170] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello. The Location Stack: A Layered Model for Location in Ubiquitous Computing. In *Proceedings of WMCSA 2002*, Callicoon, USA, June 2002.

[171] Brandon Jones and Nell Waliczek. WebXR Device API. Technical report, World Wide Web Consortium (W3C), July 2020. URL https://www.w3.org/TR/webxr/.

[172] Vipul Kashyap, Christoph Bussler, and Matthew Moran. *The Semantic Web: Semantics for Data and Services on the Web*. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-76451-9. doi: 10.1007/978-3-540-76452-6.

[173] Edward A. Lee and Thomas M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995. doi: 10.1109/5.381846.

[174] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and T. Schaub. The GeoJSON Format. RFC 7946, August 2016.

[175] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M Dennis Mickunas. MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications. In *Proceedings of Middleware 2004*, Subotica, Serbia, October 2004.

[176] Jeffrey Hightower and Gaetano Borriello. Location Systems for Ubiquitous Computing. *Computer*, 34(8), August 2001. doi: 10.1109/2.940014.

[177] Mansfield E. de Jong J. Math.js: An Advanced Mathematics Library for JavaScript. *Computing in Science & Engineering*, 20(1), January 2014. doi: 10.1109/MCSE.2018.011111122.

[178] Adam L. Davis. Akka Streams. In *Reactive Streams in Java*. Apress, Berkeley, CA, 2019. doi: 10.1007/978-1-4842-4176-9_6.

[179] Nishant Garg. *Apache Kafka*. Packt Publishing, 2013. ISBN 1782167935.

[180] Martín Abadi et al. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016)*, pages 265–283, Savannah, USA, November 2016. ISBN 978-1-931971-33-1.

[181] Marc Geilen and Twan Basten. Reactive Process Networks. In *Proceedings of EMSOFT 2004, International Conference on Embedded Software*, Pisa, Italy, September 2004. doi: 10.1145/1017753.1017778.

[182] Esteban Zimányi, Mahmoud Sakr, Arthur Lesuisse, and Mohamed Bakli. MobilityDB: A Mainstream Moving Object Database System. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases (SSTD 2019)*, Vienna, Austria, August 2019. doi: 10.1145/3340964.3340991.

[183] Web Bluetooth Community Group. Web Bluetooth Scanning. W3c working draft, W3C, March 2023. URL https://webbluetoothcg.github.io/web-bluetooth/scanning.html.

[184] Chris Currier. *Protocol Buffers*, pages 223–260. Springer International Publishing, Cham, 2022. doi: 10.1007/978-3-030-98467-0_9.

[185] Ido Green. *Web Workers: Multithreaded Programs in JavaScript*. O'Reilly Media, Inc., 2012. ISBN 978-1449322137.

[186] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11), 2000. URL https://www.drdobbs.com/open-source/the-opencv-library/184404319.

[187] Witold Litwin and Abdelaziz Abdellatif. Multidatabase Interoperability. *Computer*, 19(12):10–18, 1986. doi: 10.1109/MC.1986.1663123.

[188] María Poveda-Villalón, Alba Fernández-Izquierdo, Mariano Fernández-López, and Raúl García-Castro. LOT: An Industrial Oriented Ontology Engineering Framework. *Engineering Applications of Artificial Intelligence*, 111, May 2022. doi: 10.1016/j.engappai.2022.104755.

[189] Pierre-Yves Vandenbussche, Ghislain A. Atemezing, María Poveda-Villalón, and Bernard Vatant. Linked Open Vocabularies (LOV): A Gateway to Reusable semantic Vocabularies on the Web. *Semantic Web*, 8(3):437–452, 2016. doi: 10.3233/SW-160213.

[190] Armin Haller, Krzysztof Janowicz, Simon Cox, Danh Le Phuoc, Kerry Taylor, and Maxime Lefrançois. Semantic Sensor Network Ontology. World Wide Web Consortium, 2017. URL https://www.w3.org/TR/vocab-ssn/.

[191] Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P Avariento, Tomás J Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. UJIIndoorLoc: A New Multi-Building and Multi-Floor Database for WLAN Fingerprint-based Indoor Localization Problems. In *Proceedings of the 5th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2014)*, pages 261–270, 2014. doi: 10.1109/IPIN.2014.7275492.

[192] Joaquín Torres-Sospedra, David Rambla, Raul Montoliu, Oscar Belmonte, and Joaquín Huerta. UJIIndoorLoc-Mag: A New Database for Magnetic Field-based Localization Problems. In *Proceedings of the 6th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2015)*, pages 1–10, 2015. doi: 10.1109/IPIN.2015.7346763.

[193] Senbo Wang, Jiguang Yue, Yanchao Dong, Shibo He, Haotian Wang, and Shaochun Ning. A Synthetic Dataset for Visual SLAM Evaluation. *Robotics and Autonomous Systems*, 124:103336, 2020. doi: 10.1016/j.robot.2019.103336.

[194] Ralph Hodgson, Paul J Keller, Jack Hodges, and Jack Spivak. QUDT-Quantities, Units, Dimensions and Data Types Ontologies. https://qudt.org, 2014.

[195] Francisco Zampella, Antonio Ramón Jiménez Ruiz, and Fernando Seco Granja. Indoor Positioning Using Efficient Map Matching, RSS Measurements, and an Improved Motion Model. *IEEE Transactions on Vehicular Technology*, 64(4), 2015. doi: 10.1109/TVT.2015.2391296.

[196] ISO Central Secretary. Geographic Information – Referencing by Coordinates. Standard ISO 19111:2019, International Organization for Standardization, 2019. URL https://www.iso.org/standard/74039.html.

[197] Evandro Bernardes and Stéphane Viollet. Quaternion to Euler Angles Conversion: A Direct, General and Computationally Efficient Method. *PLOS ONE*, 17(11): e0276302, 2022. doi: 10.1371/journal.pone.0276302.

[198] Michael Compton, Payam Barnaghi, Luis Bermudez, Raul Garcia-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, et al. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. *Journal of Web Semantics*, 17, December 2012. doi: 10.1016/j.websem.2012.05.003.

[199] Zhixian Yan, Jose Macedo, Christine Parent, and Stefano Spaccapietra. Trajectory Ontologies and Queries. *Transactions in GIS*, 12:75–91, 2008. doi: 10.1111/j.1467-9671.2008.01137.x.

[200] Yingjie Hu, Krzysztof Janowicz, David Carral, Simon Scheider, Werner Kuhn, Gary Berg-Cross, Pascal Hitzler, Mike Dean, and Dave Kolas. A Geo-ontology Design Pattern for Semantic Trajectories. In *Proceedings of the International Conference on Spatial Information Theory (COSIT 2013)*, pages 438–456, 2013. doi: 10.1007/978-3-319-01790-7_24.

[201] Isaac Skog, Gustaf Hendeby, and Fredrik Gustafsson. Magnetic Odometry: A Model-Based Approach Using a Sensor Array. In *Proceedings of the International Conference on Information Fusion (FUSION 2018)*, pages 794–798, 2018. doi: 10.23919/ICIF.2018.8455430.

[202] Sebastian Thrun. Simultaneous Localization and Mapping. In *Robotics and Cognitive Approaches to Spatial Mapping*, pages 13–41. Springer, 2007. doi: 10.1007/978-3-540-75388-9_3.

[203] Amelie Gyrard, Soumya Kanti Datta, Christian Bonnet, and Karima Boudaoud. Cross-Domain Internet of Things Application Development: M3 Framework and Evaluation. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud 2015)*, August 2015. doi: 10.1109/FiCloud.2015.10.

[204] Andrew Iliadis, Amelia Acker, Wesley Stevens, and Sezgi Başak Kavakli. One Schema to Rule them All: How Schema.org Models the World of Search. *Journal of the Association for Information Science and Technology*, 2023. doi: 10.1002/asi.24744.

[205] Alberto Hernández Chillón, Diego Sevilla Ruiz, Jesus García Molina, and Severino Feliciano Morales. A Model-Driven Approach to Generate Schemas for Object-Document Mappers. *IEEE Access*, 7:59126–59142, 2019. doi: 10.1109/ACCESS.2019.2915201.

[206] Noel De Martin. Soukai, 2024. URL https://soukai.js.org/. Accessed on May 2024.

[207] O.team. Linked Data Objects (LDO), 2024. URL https://ldo.js.org/. Accessed on May 2024.

[208] Ken Wenzel. KOMMA: An Application Framework for Ontology-Based Software Systems. *Semantic Web–Interoperability, Usability, Applicability*, 1:1–10, 2010.

[209] Vincent Tunru. rdf-namespaces, 2021. URL https://gitlab.com/vincenttunru/rdf-namespaces. Accessed on June 2021.

[210] Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT Queries in SPARQL. In Marcelo Arenas and Martín Ugarte, editors, *Proceedings of the 18th International Conference on Database Theory (ICDT 2015)*, volume 31, pages 212–229, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi: 10.4230/LIPIcs.ICDT.2015.212.

[211] L Cabral, J Haucap, G Parker, G Petropoulos, T Valletti, and M Alstyne. *The EU Digital Markets Act: A Report From a Panel of Economic Experts*. Publications Office of the European Office, 2021. doi: 10.2760/139337.

[212] René Hansen, Rico Wind, Christian S Jensen, and Bent Thomsen. Seamless Indoor/Outdoor Positioning Handover for Location-based Services in Streamspin. In *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM 2009)*, May 2009. doi: 10.1109/MDM.2009.39.

[213] Wout Slabbinck, Ruben Dedecker, Sindhu Vasireddy, Ruben Verborgh, and Pieter Colpaert. Linked Data Event Streams in Solid LDP Containers. In *Proceedings of the 8th Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW 2022)*, volume 3339, 2022.

[214] Arnaud J. Le Hors and Steve Speicher. The Linked Data Platform (LDP). In *Proceedings of the 22nd International Conference on World Wide Web (WWW 2013)*, page 1–2. ACM New York, NY, USA, 2013. doi: 10.1145/2487788.2487790.

[215] Dwight Van Lancker, Pieter Colpaert, Harm Delva, Brecht Van de Vyvere, Julián Rojas Meléndez, Ruben Dedecker, Philippe Michiels, Raf Buyle, Annelies De Craene, and Ruben Verborgh. Publishing Base Registries as Linked Data Event Streams. In *Proceedings of the 21st International Conference on Web Engineering (ICEW 2021)*, pages 28–36, Cham, 2021. Springer International Publishing. doi: 10.1007/978-3-030-74296-6_3.

[216] Pieter Colpaert. Building Materializable Querying Interfaces With the TREE Hypermedia Specification. In *Proceedings of MEPDaW@ISWC*, pages 8–18, 2022. URL https://treecg.github.io/paper-materializable-interfaces/.

[217] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*, April 2014. URL https://ceur-ws.org/Vol-1184/ldow2014_paper_04.pdf.

[218] Sarven Capadisli, Amy Guy, Christoph Lange, Sören Auer, Andrei Sambra, and Tim Berners-Lee. Linked Data Notifications: A Resource-centric Communication

Protocol. In *Proceedings of the 14th International Conference on The Semantic Web (ESWC 2017)*, Portorož, Slovenia, 2017. doi: 10.1007/978-3-319-58068-5_33.

[219] Ruben Taelman and Ruben Verborgh. Link Traversal Query Processing Over Decentralized Environments with Structural Assumptions. In *Proceedings of the 22nd International Semantic Web Conference (ISWC 2023)*, pages 3–22. Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-47240-4_1.

[220] HS Rajashree et al. Indoor Localization using BLE Technology. *International Journal of Engineering Research & Technology (IJERT)*, 6(13), 2018. ISSN 2278-0181.

[221] Markus Koühne and Jürgen Sieck. Location-Based Services With iBeacon Technology. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS 2014)*, pages 315–321, 2014. doi: 10.1109/AIMS.2014.58.

[222] Apple Inc. Proximity Beacon Specification. Specification, Apple Inc., 2015. URL https://developer.apple.com/ibeacon/. Accessed on November 2022.

[223] Rodrigo VM Pereira et al. A Digital Implementation of Eddystone Standard Using IBM 180nm Cell Library. In *Proceedings of the 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2017. doi: 10.1109/SBESC.2017.28.

[224] Daeil Seo and Byounghyun Yoo. Interoperable Information Model for Geovisualization and Interaction in XR Environments. *IJGIS*, 34(7), 2020. doi: 10.1080/13658816.2019.1706739.

[225] Bluetooth SIG Inc. Indoor Positioning Service. Specification, Bluetooth SIG Inc., 2015. URL https://bluetooth.com/specifications/specs/indoor-positioning-service-1-0/.

[226] Hamza Soganci, Sinan Gezici, and H Vincent Poor. Accurate Positioning in Ultra-Wideband Systems. *IEEE Wireless Communications*, 18(2), 2011. doi: 10.1109/MWC.2011.5751292.

[227] Jeff Z. Pan. Resource Description Framework. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 71–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-540-92673-3_3.

[228] Oscar Corcho et al. A High-Level Ontology Network for ICT Infrastructures. In *Proceedings of the 20th International Semantic Web Conference (ISWC 2021)*, 2021. doi: 10.1007/978-3-030-88361-4_26.

[229] Rudy Arthur. A Critical Analysis of the What3Words Geocoding Algorithm. *PLOS ONE*, 18(10):1–13, 10 2023. doi: 10.1371/journal.pone.0292491.

[230] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the 1st International Workshop of Peer-to-Peer Systems (IPTPS 2002)*, pages 53–65. Springer Berlin Heidelberg, 2002. doi: 10.1007/3-540-45748-8_5.

[231] Quanhao Lin, Ruonan Rao, and Minglu Li. DWSDM: A Web Services Discovery Mechanism Based on a Distributed Hash Table. In *Proceedings of the 5th International Conference on Grid and Cooperative Computing Workshops*, pages 176–180, 2006. doi: 10.1109/GCCW.2006.36.

[232] Yahya Hassanzadeh-Nazarabadi, Sanaz Taheri-Boshrooyeh, Safa Otoum, Seyhan Ucar, and Öznur Özkasap. DHT-based Communications Survey: Architectures and Use Cases. *arXiv preprint arXiv:2109.10787*, 2021.

[233] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 739–752, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394208. doi: 10.1145/3544216.3544232.

[234] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: A Public DHT Service and Its Uses. *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, August 2005. ISSN 0146-4833. doi: 10.1145/1090191.1080102.

[235] Martin Atkins, Will Norris, Chris Messina, Monica Wilkinson, Rob Dolin, and James Snell. Activity Streams 2.0. W3C Editor's Draft, World Wide Web Consortium (W3C), May 2017. URL https://www.w3.org/TR/activitystreams-core/.

[236] Maxim Van de Wynckel. Putting an Artificial Brain in my Sumo Robot, June 2019. URL https://medium.com/@Maximvdw/putting-an-artificial-brain-in-my-sumo-robot-9bdb5e8671d.

[237] Ngewi Fet, Marcus Handte, and Pedro José Marrón. A Model for WLAN Signal Attenuation of the Human Body. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2013)*, Zurich, Switzerland, September 2013. doi: 10.1145/2493432.2493459.

[238] Adriano Moreira, Maria João Nicolau, Filipe Meneses, and António Costa. Wi-Fi Fingerprinting in the Real World: RTLS@UM at the EvAAL Competition. In *Proceedings of the 6th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2015)*, Calgary, Canada, July 2015. doi: 10.1109/IPIN.2015.7346967.

[239] Yassine Bouagaz. Anonymous Contact Tracing Fusion. Master's thesis, Vrije Universiteit Brussel, Elsene, Belgium, 2021. Available at https://researchportal.vub.be/en/publications/anonymous-contact-tracing-fusion-2.

[240] Joachim Van Herwegen and Ruben Verborgh. Granular Access to Policy-Governed Linked Data via Partial Server-Side Query. In *Proceedings of the 21st Extended Semantic Web Conference (ESWC 2024)*, May 2024. doi: 10.1007/978-3-031-78952-6_52.

[241] Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: a Modular SPARQL Query Engine for the Web. In *Proceedings of the 17th International Semantic Web Conference (ISWC 2018)*, October 2018. doi: 10.1007/978-3-030-00668-6_15.

[242] Olaf Hartig and Johann-Christoph Freytag. Foundations of Traversal Based Query Execution Over Linked Data. In *Proceedings of Hypertext'21*, June 2012. doi: 10.1145/2309996.2310005.

[243] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic Generation and Detection of Highly

Reliable Fiducial Markers Under Occlusion. *Pattern Recognition*, 47(6), 2014. doi: 10.1016/j.patcog.2014.01.005.

[244] Tobias Käfer, Andreas Harth, and Sebastién Mamessier. Towards Declarative Programming and Querying in a Distributed Cyber-Physical System: The i-VISION Case. In *Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data 2016)*, Vienna, Austria, 2016. doi: 10.1109/CPSData.2016.7496418.

[245] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing Linked Data Quality Assessment. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*, Sydney, Australia, 2013. doi: 10.1007/978-3-642-41338-4_17.

[246] Weimin Mou, Frank Biocca, Charles B Owen, Arthur Tang, Fan Xiao, and Lynette Lim. Frames of Reference in Mobile Augmented Reality Displays. *Journal of Experimental Psychology Applied*, 10(4), 2004. doi: 10.1037/1076-898X.10.4.238.

[247] Wallace S. Lages and Doug A. Bowman. Walking With Adaptive Augmented Reality Workspaces: Design and Usage Patterns. In *Proceedings of the 24th International Conference on Intelligent User Interfaces (IUI 2019)*, 2019. doi: 10.1145/3301275.3302278.

[248] Michail Kalaitzakis, Brennan Cain, Sabrina Carroll, Anand Ambrosi, Camden Whitehead, and Nikolaos Vitzilaios. Fiducial Markers for Pose Estimation: Overview, Applications and Experimental Comparison of the ARTag, AprilTag, ArUco and STag Markers. *Journal of Intelligent & Robotic Systems*, 101, 2021. doi: 10.1007/s10846-020-01307-9.

[249] Juan D. Gutiérrez, Antonio R. Jiménez, Fernando Seco, Fernando J. Álvarez, Teodoro Aguilera, Joaquín Torres-Sospedra, and Fran Melchor. GetSensorData: An Extensible Android-based Application for Multi-Sensor Data Registration. *SoftwareX*, 19:101186, 2022. ISSN 2352-7110. doi: 10.1016/j.softx.2022.101186.

[250] Janne Parkkila, Filip Radulovic, Daniel Garijo, María Poveda-Villalón, Jouni Ikonen, Jari Porras, and Asunción Gómez-Pérez. An Ontology for Videogame Interoperability. *Multimedia Tools and Applications*, 76(4):4981–5000, February 2017. doi: 10.1007/s11042-016-3552-6.

[251] Robert Shirey. RFC 4949: Internet Security Glossary, Version 2, 2007. doi: 10.17487/RFC4949.

[252] Sarven Capadisli, Tim Berners-Lee, and Henry Story. Web Access Control. Specification, W3C Solid Community Group, June 2024. URL https://solid.github.io/web-access-control-spec/.

[253] Harshvardhan J. Pandit, Beatriz Esteves, Georg P. Krog, Paul Ryan, Delaram Golpayegani, and Julian Flake. Data Privacy Vocabulary (DPV) – Version 2, 2024. doi: 10.48550/arXiv.2404.13426.

[254] Christian Esposito, Ross Horne, Livio Robaldo, Bart Buelens, and Elfi Goesaert. Assessing the Solid Protocol in Relation to Security and Privacy Obligations. *Information*, 14(7), 2023. ISSN 2078-2489. doi: 10.3390/info14070411.

[255] Omid Mirzamohammadi, Kristof Jannes, Laurens Sion, Dimitri Van Landuyt, Aysajan Abidin, and Dave Singelée. Security and Privacy Threat Analysis for Solid. In *Proceedings of the IEEE Secure Development Conference (SecDev 2023)*, pages 196–206, 2023. doi: 10.1109/SecDev56634.2023.00033.

[256] Ahmed Abid, Alexey Medvedev, Ali Hassani, Franck Le Gall, Giuseppe Tropea, Juan Antonio Martinez, Lindsay Frost, and Martin Bauer. Guidelines for Modelling with NGSI-LD, January 2021.

[257] J Odmins, K Slics, R Fenuks, E Linina, K Osmanis, and I Osmanis. Comparison of Passive and Active Fiducials for Optical Tracking. *Latvian Journal of Physics and Technical Sciences*, 59(5), 2022. doi: 10.2478/lpts-2022-0040.

[258] Martin Kaltenbrunner and Ross Bencina. reacTIVision: A Computer-Vision Framework for Table-based Tangible Interaction. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI 2007)*, Baton Rouge, USA, February 2007. doi: 10.1145/1226969.1226983.

[259] Lilian Calvet, Pierre Gurdjos, Carsten Griwodz, and Simone Gasparini. Detection and Accurate Localization of Circular Fiducials Under Highly Challenging Conditions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 2016. URL https://openaccess.thecvf.com/content_cvpr_2016/html/Calvet_Detection_and_Accurate_CVPR_2016_paper.html.

[260] Charles E Kahn Jr, Curtis P Langlotz, David S Channin, and Daniel L Rubin. Informatics in Radiology: An Information Model of the DICOM Standard. *Radiographics*, 31(1), 2011. doi: 10.1148/rg.311105085.

[261] ISO Central Secretary. Information technology – Automatic Identification and Data Capture Techniques. Standard ISO 18004:2015, International Organization for Standardization, 2021. URL https://www.iso.org/standard/62021.html.

[262] Evalds Urtans and Agris Nikitenko. Active Infrared Markers for Augmented and Virtual Reality. *Engineering for Rural Development*, 9:10, 2016. URL https://api.semanticscholar.org/CorpusID:29944026.

[263] Edwin Olson. AprilTag: A Robust and Flexible Visual Fiducial System. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2011)*, Shanghai, China, May 2011. doi: 10.1109/ICRA.2011.5979561.

[264] Francely Franco Bermudez, Christian Santana Diaz, Sheneeka Ward, Rafael Radkowski, Timothy Garrett, and James Oliver. Comparison of Natural Feature Descriptors for Rigid-Object Tracking for Real-Time Augmented Reality. In *Proceedings of the 34th Computers and Information in Engineering Conference (ASME 2014)*, Buffalo, USA, August 2014. doi: 10.1115/DETC2014-35319.

Index

About the Author

Maxim Van de Wynckel was born in Belgium in 1995. He obtained a Bachelor of Applied Computer Science from the Erasmushogeschool Brussel in 2016, graduating with great distinction. He subsequently earned a Master of Science in Applied Computer Science from the Vrije Universiteit Brussel in 2019, also with great distinction. His Master's thesis, titled *Indoor Navigation by Centralised Tracking*, was supervised by Prof. Dr. Olga De Troyer. During his Master's studies, he identified the fragmented landscape of indoor navigation systems and the challenges arising from their heterogeneity.

Following his Master's studies, he pursued a PhD at the Web & Information Systems Engineering (WISE) Lab under the supervision of Prof. Dr. Beat Signer. His doctoral research focused on the interoperability of these systems to facilitate their deployment. This work on interoperable and discoverable indoor positioning systems resulted in eight peer-reviewed publications, nine presentations at international conferences, and media coverage.

During his time at the Vrije Universiteit Brussel, Maxim played a key role in the development of OpenHPS, an open-source hybrid positioning system released in 2020, which constitutes a major contribution to this dissertation with many established extensions. In 2023, he introduced the SemBeacon Bluetooth Low Energy specification, enabling semantic beacons to describe and track people, objects, and environments.

In addition to his research, Maxim served as a teaching assistant for several Bachelor's and Master's courses, including Web Technologies, Databases, Next Generation User Interfaces, Information Visualisation, Advanced Topics in Big Data, and Open Information Systems. He supervised four Bachelor's thesis students and six Master's thesis students, all within his research domain.

Personal website:	`https://maximvdw.be`
Academic Curriculum Vitae:	`https://maximvdw.be/cv/academic/`
Dissertation website:	`https://phd.maximvdw.be`