Deliverable

D3.2 IoT stack and API specifications v2

Onboarding IoT data in the DUET platform

Project Acronym:	DUET	
Project title:	Digital Urban European Twins	
Grant Agreement No.	870697	
Website:	www.digitalurbantwins.eu	
Version:	1.0	
Date:	04.06.2021	
Responsible Partner:	imec	
Contributing Partners:	OASC, AIV	
Reviewers:	ATC (T. Dalianis) Thomas Adolphi (VCS) TNO (Walter Lohman) Yannis Charalabidis Pieter Morlion	
Dissemination Level:	Public	X
	Confidential – only consortium members and European Commission	

This project has received financial support from the European Union's Horizon 2020 Programme under grant agreement no. 870697

Revision History

Revision	Date	Author	Organization	Description
0.1	20.03.2021	Philippe Michiels	imec	Structure
0.2	26.04.2021	Sigve Vermandere	imec	Part 4.1 & 4.2
0.3	01.05.2021	Philippe Michiels	imec	Intro & D3.1 recap
0.4	02.05.2021	Jan Willem	imec	Part 2.4 & 5.2
0.5	02.05.2021	Dwight Van Lancker	AIV	Part 3.2
0.6	03.05.2021	Gert De Tant	OASC	Added Data Lake House for IoT & context data
0.7	21.05.2021	Gert De Tant	OASC	Added additional info regarding iot and context data + Security
0.8	24.05.2021	Philippe Michiels	imec	Ready for review version
0.9	31.05.2021	Pieter Morlion	MORE LION	External review
1.0	02.06.2021	Philippe Michiels	imec	Final version

Table of Contents

Executive Summary 6
1 Introduction 7
1.1 Objectives 7
1.2 Recap of delivery 3.1 8
1.3 Document structure 8
1.4 Roadmap 9
1.5 Relation to other documents 9
2 Smart Data Management 10
2.1 IoT Data, Big Data and Smart Data Management 10
From Big Data to Smart Data 10
Smart data principles for Urban Digital Twins 11
2.2 Digital Twins as Smart Data Management Components 12
2.3 Federated Asset Broker 15
2.4 Onboarding IoT & Context Data 16
2.5 Data Lake House for IoT & context data 17
Terminology 18
Raw Data in Data Lakes 18
Data Enrichment 18
Data Lakes and Models 19
3 The IoT standards landscape 20
3.1 Existing standards 20
OGC SensorThings API 21
FIWARE 22
Web of Things 23
Semantic Sensor Network Ontology 24
Observation and Measurements 24
Smart Applications Reference 24
3.2 Interoperability of IoT and Geospatial Data 26
Different geospatial urban data sources 26
Integration of geospatial data web services and sensor web services 26

Geospatial data from IoT devices 26
Geospatially connecting data 27
4 Virtualizing IoT data 28
4.1 Reference refined data schema 28
4.2 Mapping frameworks 29
4.3 Mapping in DUET 30
5 Interactions, Models and IoT Context Data 32
5.1 Model serving 32
5.2 Interactions & context serving 32
Cases and Scenarios 32
City Context Data 33
Approach 1: a full copy 33
Approach 2: tracking changes 35
5.3 What-If Scenario Data flow 36
6 Privacy & Security 38
6.1 IoT & Context Data Security 38
6.2 Personal IoT & Context Data Handling 39
7 Conclusions & further work 42
8 References 43

Figures

Figure 1: From big data to smart data: storing data in a fit for purpose way, applying the principles of technical and semantic interoperability and publishing the data so that other users can find it and use it for their own cases. 11
Figure 2: Conceptual architecture for Urban Digital Twins: The figure maps DT components onto the VLOCA conceptual architecture. This allows building a federated network of Digital Twin Infrastructure. 13
Figure 3: Federated asset broker: By interconnecting the brokers, multiple digital twins can form a federated network in which all assets can be shared as if they were residing in the local digital twin's broker. 15
Figure 4: The LDES publication pipeline: By means of an adapter, the source system publishes events according to the Linked Data Event Stream (LDES) standard. 16
Figure 5: Raw data is ingested as-is in a first step. 18
Figure 6: Raw data is enriched with context data from a meta-store. 19
Figure 7: Categorizing IoT standards from more technical specific (left) to more business specific (right) 20
Figure 8: Table describing SensorThings' data model can be mapped on the data model of Observation and Measurements (O&M) 21
Figure 9: The main elements in the NGSI data model are context entities, attributes and metadata 22
Figure 10: layered model to define the NGSI-LD information model 22
Figure 11: Image showing that the Semantic Sensor Network ontology includes SOSA as its core vocabulary and can be mapped on Observation and Measurements by using the O&M alignment module 24
Figure 12: Mapping NGSI-LD to SAREF (Source: ETSI GS CIM 006) 25
Figure 13: The OGC data standards landscape 26
Figure 14: Virtualizing data sources through mapping in DUET: When data is streamed across the DUET message streaming platform, it can be mapped on the fly using (predefined) mappings. As such both the raw data stream and the mapped one are available to all users. Currently, WASM is implemented as a powerful mechanism. But maintaining mapping code can be complex. 28
Figure 15: SHACL example of Air Quality Index 29
Figure 16: The organization of cases and scenarios in Digital Twins for supporting what-if scenarios. 33
Figure 17: A conceptual picture of a context graph where hypothetical situations can be shaped and saved. 34
Figure 18: A basic context serving scenario 35
Figure 19: The what-if scenario data flow with model interactions 36
Figure 20: The ISMS framework from INESA 38
Figure 21: An attribute based protection example case 39
Figure 22: SOLID Web ACL layer for IoT data. 39
Figure 23: A personal data operator can act as a middle-man for exposing personal data securely and efficiently. 41

Executive Summary

After implementing the closed beta version of DUET, we have gained valuable insight into the way digital twins can or should work. This deliverable re-iterates some of the subjects discussed in Deliverable D3.1 and also touches on some other subjects.
In this document, we discuss how Smart Data Management principles are essential for realizing digital twins. A robust data management approach is essential for digital twins. Having only partial data or data whose nature is not well understood will not lead to the quality necessary for evidence-based policies.
Also, the standards landscape for IoT and related data sources has been scrutinized. Understanding this landscape can help define the minimal set of tools needed to obtain the level of integration that we need to onboard the available data sources.
The variety of data standards also shows that we needed to pivot our view on how to deal with data formats. Instead of defining a single internal data model and map every data source onto one schema, we evolved towards a data virtualization approach. Mapping there can be done at any time in the data pipeline, making for a more flexible data architecture better suited to accommodate all kinds of tools and clients.
Next, we have looked at the role of context data in what-if scenarios and presented a high-level concept of dealing with city context data in the case of what-if scenarios while remaining consistent with the principles of DUET.
Finally, we have looked at security in the context of DUET and IoT data. Web Access Control Lists are an existing concept that is used to control access to web data. We propose to look more closely at how this technology can be incorporated into the DUET security architecture.
Although out of scope for DUET, personal data will become a relevant input vector for digital twins. We briefly discussed how this kind of data could be used in digital twins while remaining vigilant regarding personal privacy and complying with GDPR rules.

The next step in the implementation of DUET is to take the lessons learned and start integrating them into the DUET architecture. An integration phase is now needed to consolidate the components into a robust basic digital twin solution (see 1.4 Roadmap). Aside from that, there are plenty of areas to continue our research: Linked Data Event Streams for connecting evolving data sources and implementing City Context Data services for interaction support. But also security and access control remain to be integrated at the data source level.

1 Introduction

This Deliverable is a continuation of deliverable D3.1. Some of the questions that were left unanswered in D3.1 are answered here. Furthermore, other concerns regarding IoT data in digital twins are addressed as well. For example, we look at privacy and security concerns for IoT data and we discuss how IoT context data plays an essential role in what-if scenarios.

As such the scope of this deliverable is a bit broader than the one of D3.1 as we aim to identify the data architecture of digital twins. This starts with onboarding the data, transforming and processing the data and ends with supporting interactions in the virtual world, flowing back to the data sources, the simulation models and the visualizations.

More specifically we will cover:

1. Smart data management as an enabler for digital twins
2. The IoT data standards landscape
3. Data virtualization as a key approach is achieve the necessary data flexibility
4. IoT data in relation to models and interaction support
5. Privacy and security for IoT data

We discuss the most important insights that we have gathered from our implementation journey so far, and strengthen our design with new principles and components.

1.1 Objectives

In Urban Digital Twins the worlds of big data and Internet of Things come together to form a new one. The challenge is to combine proven strategies and technologies of these two worlds to work together.

Some challenges follow from the fact that data sources of different providers and origin need to be combined. Producing reliable results from processing of combined data in computational models is a nontrivial task. After all, there is little or no control over the properties of that data in terms of meaning, structure, accuracy, etc.

In the world of big data on the other hand, the focus is mostly on the sheer size of the data. Typically, data sets are huge and special techniques have to be used to ensure scalable data processing. Scalable storage systems called data lake houses [1] (an evolution of data warehouses and data lakes) are the foundation for robust data processing. They are combined with efficient data transfer technologies such as AVRO[1], Parquet[2] or ORC[3] to facilitate an efficient data flow towards data processing and analysis systems. When moving the data itself becomes too big of a challenge, new deployment technologies such as containers and portable algorithms can be used to move the model to the data instead.

This deliverable reflects the DUET vision of combining aspects of both worlds to build open, scalable urban digital twin platforms that enable the reuse of data, simulation models and clients. It also explains the lessons learned so far and attempts to plot a way forward.

[1] see Apache Avro: https://avro.apache.org/
[2] see Apache Parquet: https://parquet.apache.org
[3] see Apache ORC: https://orc.apache.org/

1.2 Recap of delivery 3.1

We try to deliver on the promise made in the first version of this document to dig deeper in some open issues.

In Deliverable D3.1, we discussed typical IoT stacks and solutions along with some commonly used architectures and standards. We discussed the different types of IoT data: event data, time series data and context data. We also briefly touched on other data sources such as geospatial data and reference data sources.
In this deliverable we build upon that to look at how we can uniformly deal with the management of such data by extending the concept of a data catalogue and introducing smart data management (SDM).

Deliverable 3.1 also introduced the concept of receptors to make this data accessible to other digital twin components. This was the basis for the DUET data broker architecture which was extensively discussed in another deliverable D3.8.
In this deliverable we continue the journey and look at how standards, the data broker and universal mappings can come together in the notion of data virtualization. We also discuss how these data sources can be queried in a universal way through DUET's data services (cfr. DUET query language).

Another question that was left unanswered was the problem of providing interaction support in DUET. We propose a conceptual answer to supporting interactions, which are a key to implementing what-if scenarios.

Finally we also take a closer look at challenges around geographical data and more specifically how geographical data can be matched with other data sources. The lack of references, unified identifiers and uniform coordinate systems can render the task of combining such data sources very challenging.

1.3 Document structure

In this document we look at how we can connect data, notably IoT data to the DUET system and how other digital twin components such as visualization clients and simulation models can interact through standard APIs with the DUET core.

In section 2, we briefly present the insights we have gathered around data management in the context of digital twins. We discuss key concepts such as refined data schema and data time travel. We also map the DUET components onto a conceptual smart data management architecture which serves among others as a foundation for Open Smart City architectures. We discuss how this conceptual architecture accommodates data storage and how it relates to the duet architecture.

In Section 3 we discuss insights gathered around the interoperability and data IoT-related standards. We look at common strategies for connecting IoT data and discuss interoperability of Geographical data sources.

Section 4 explains the difference between a data integration approach where all data is mapped onto a common data model, and a data virtualization approach where refined schema information is used to map between input and output data formats. We explain why we are looking at the latter approach for DUET.

Eventually the data comes together in models and visualization clients. Section 5 explains how they can connect to the DUET data services to fetch the data and - in return - publish their results.

Finally we discuss the security and privacy concerns around IoT data in Section 6.

1.4 Roadmap

For the next steps in the development of DUET, an integration phase is needed to consolidate the components into a robust solution.

We identify the following primary goals:

- Integrate the data catalogue (i.e., show Smart Data Management for DTs - see 2Smart Data Management) and have all components use it to access data and relay messages (i.e., show DUET Message Broker concept)
- Implement the model catalogue (i.e., demonstrate the DUET Model Serving - see 5.1 Model serving) and integrate it into the UI, allow users to run a model from the UI
- Implement an interaction service to properly support What-If-scenarios (demonstrate DUET scenarios - see 5.2 Interactions & context serving)
- Allow models to publish their results as a data source (prove support for Data Lakes in DUE - see 2.5 Data Lake House for IoT & context data)

Secondary goals are identified as follows:

- Further develop the data virtualization approach (4 Virtualizing IoT data) and figure out ways of simplifying mapping between application profiles and IoT standards (see 3 The IoT standards landscape).
- Build a knowledge graph system on top of the data virtualization approach
- Look at more robust and scalable ways to onboard big IoT data (see 2.4 Onboarding IoT & Context Data)

1.5 Relation to other documents

This deliverable builds upon **D3.1 IoT stack and API specifications v1** and adds the lessons learned so far. It also discusses the relationship between IoT data and the interaction framework provided by DUET which was first introduced in **D3.3 Smart City domains, models and interaction frameworks v1.** Additionally, it discusses how interactive clients and models interact with each other and discusses the model serving pattern, whose foundations have been discussed earlier in **D3.5 Cloud Design for Model Calibration and Simulation.** Furthermore, this deliverable discusses existing IoT standards, some of which have already been discussed in detail in **D3.6 OSLO Extensions for the Digital Twin**.

2 Smart Data Management

2.1 IoT Data, Big Data and Smart Data Management

In the introduction we explained how the challenges of big data and IoT data come together in Urban Digital Twins. One of the key differences between typical big data architectures and smart cities is a consequence of their context:

- Big data systems more often than not have a common enterprise focus. This implies that the data infrastructure, data sources, data formats but also the computational infrastructure are under the governance of a central entity. Data and models can be kept in proximity, providing an environment that is optimized for Big Data processing.
- In smart city ecosystems this is usually not the case. Data, models, clients and tools can come from a myriad of providers without any form of central governance. There is no central data management or storage and the computational infrastructure is distributed and fragmented. Making sure that the data and models can connect efficiently are challenges on top of the common ones around interoperability.

In order to bridge the gap, the Flemish Open City Architecture (= VLOCA) [2] provides a conceptual federated architecture for building scalable Open City Platforms as systems of systems connected through federation. The architecture can be applied to decision support systems such as Digital Twins, as we will show by mapping the DUET components onto the architecture.

From Big Data to Smart Data

VLOCA also discusses how to bridge the gap between the IoT data ecosystem and big data. Big data typically focuses on the different V's of Big Data[4]:

*Big data technologies evolved with the prime intention to capture, store, and process the semi-structured and unstructured (**variety**) data generated with high speed (**velocity**), and huge in size (**volume**). Later, these tools and technologies were explored and used for handling structured data also but preferable for storage. Eventually, the processing of structured data was still kept as optional, either using big data or traditional RDBMSs. This helps in analysing data towards effective usage of the hidden insights exposed from the data collected via social media, log files, sensors, etc. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.*

Other characteristics include: Veracity, Value, Variability, Exhaustivity, Relational, Extensional and Scalability.

In IoT and Smart City platforms, the focus is rightfully on the FAIR[5] principles: **findability**, **accessibility, interoperability** and **reusability**. It is of course essential to combine both these principles in the context of smart cities.

[4] See wikipedia on Big Data, section characteristics. (Big data - Wikipedia)
[5] See wikipedia in FAIR data. FAIR data - Wikipedia

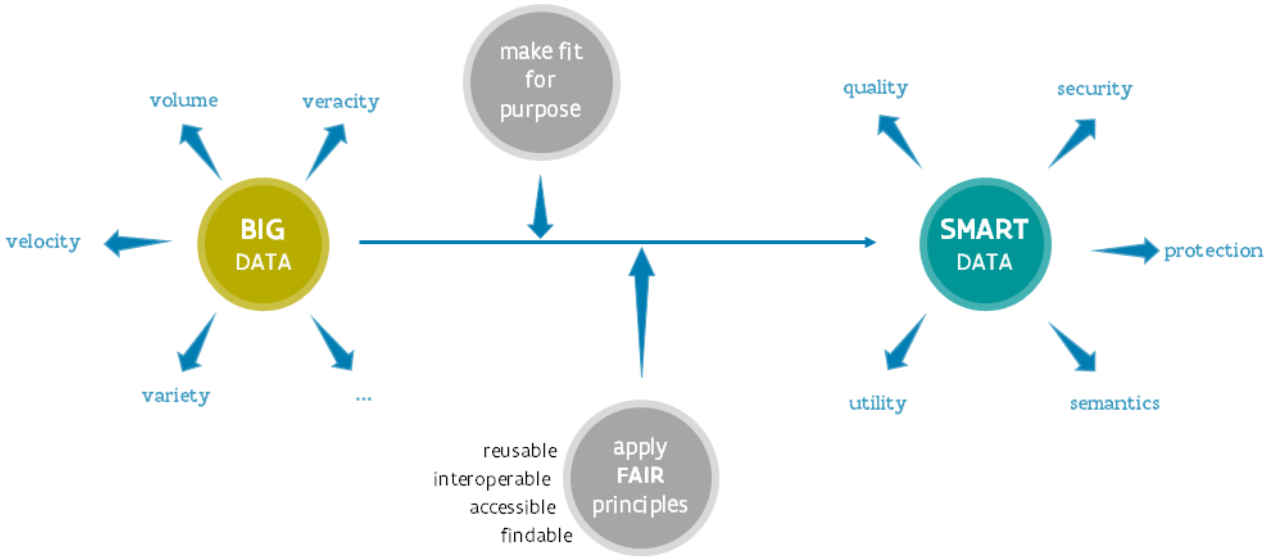

Figure 1: From big data to smart data: storing data in a fit for purpose way, applying the principles of technical and semantic interoperability and publishing the data so that other users can find it and use it for their own cases.

Smart data principles for Urban Digital Twins

Smart data management concerns many principles, below we highlight the most relevant ones for Urban Digital Twin design.

Principle 1: *Refined Data Schema*
A refined data schema provides a full and unambiguous specification for the data elements in a dataset by combining a data model, domain specific data constraints and semantics in order to facilitate full technical and semantic interoperability.
Data standards such as the Fiware Smart Data Models and OGC Semantic Sensor Network typically provide a refined schema that allows to create predefined mapping from and to their standards.

A key challenge in digital twins is to combine different data sources. Having access to a refined schema can help a great deal in understanding the data and defining mappings across schemas and vocabularies.

Principle 2: *Asset Governance*
Asset governance is the ability to manage all relevant aspects of assets (data sources/sets, models, mappings) such as:

- business and data vocabularies
- data schema
- metadata (characteristics of big data, background data, excerpts and examples, …)
- data ownership
- data access
- …

Asset governance is instrumental to data brokering, data management, data sovereignty and data privacy.

From the definition it is clear that if we want to support exchanging data sets and sharing simulation models and digital twin clients, we need to cover Asset Governance.

Principle 3: *Data Time Travel*
Evidence based policies are the key stone of the DUET project. Running experiments in hypothetical scenarios can allow city planners to better understand the impact of proposed policies. The requirement this imposes on digital twins and the data on which they rely are often underestimated.
IoT stacks often focus on keeping a full history of measurements. But having only the measurements without providing any context is not sufficient. Imagine for instance trying to understand water quality measurement results without a history of swapping sensors, cleaning them, recalibration, … So keeping a full track of context history is equally important. This is already reasonably well understood.

Note
What is not often realized is that any data source that is subjected to change over time. This includes the context. For example, when doing traffic experiments, a full history of the road network needs to remain accessible besides the history of the measurements. Maybe even the full history of meteo data as well. If not, it is impossible to reliably reproduce the outcome of an experiment after the fact, which is a key requirement when the outcome is used for policy making.

Usually this consideration is made too late, making many data sets of limited use. This observation is one of the reasons for creating the LDES (Linked Data Event Streams) specification [3][4]. LDES presents a more holistic approach towards publishing the full history of both fast moving data such as sensor data streams as slow moving data such as data in base registries.

When publishing data of any kind, keeping the full history must be considered as a good practice[6].

2.2 Digital Twins as Smart Data Management Components

The VLOCA conceptual architecture divides the technical domain into disjoint areas of concern that accommodate decoupled components fulfilling the required capabilities. The idea is to avoid black box systems spanning multiple of these areas. This would threaten interoperability, reusability (of data and systems) and come with the risk of lock-in. We discuss the different areas along with the digital twin components they can harbour.

[6] This requires implementing smart data management which is non trivial. Since DUET has no control over onboarded data, this may be difficult to ensure. It will be a necessity nevertheless when policy is based on data driven experiments.

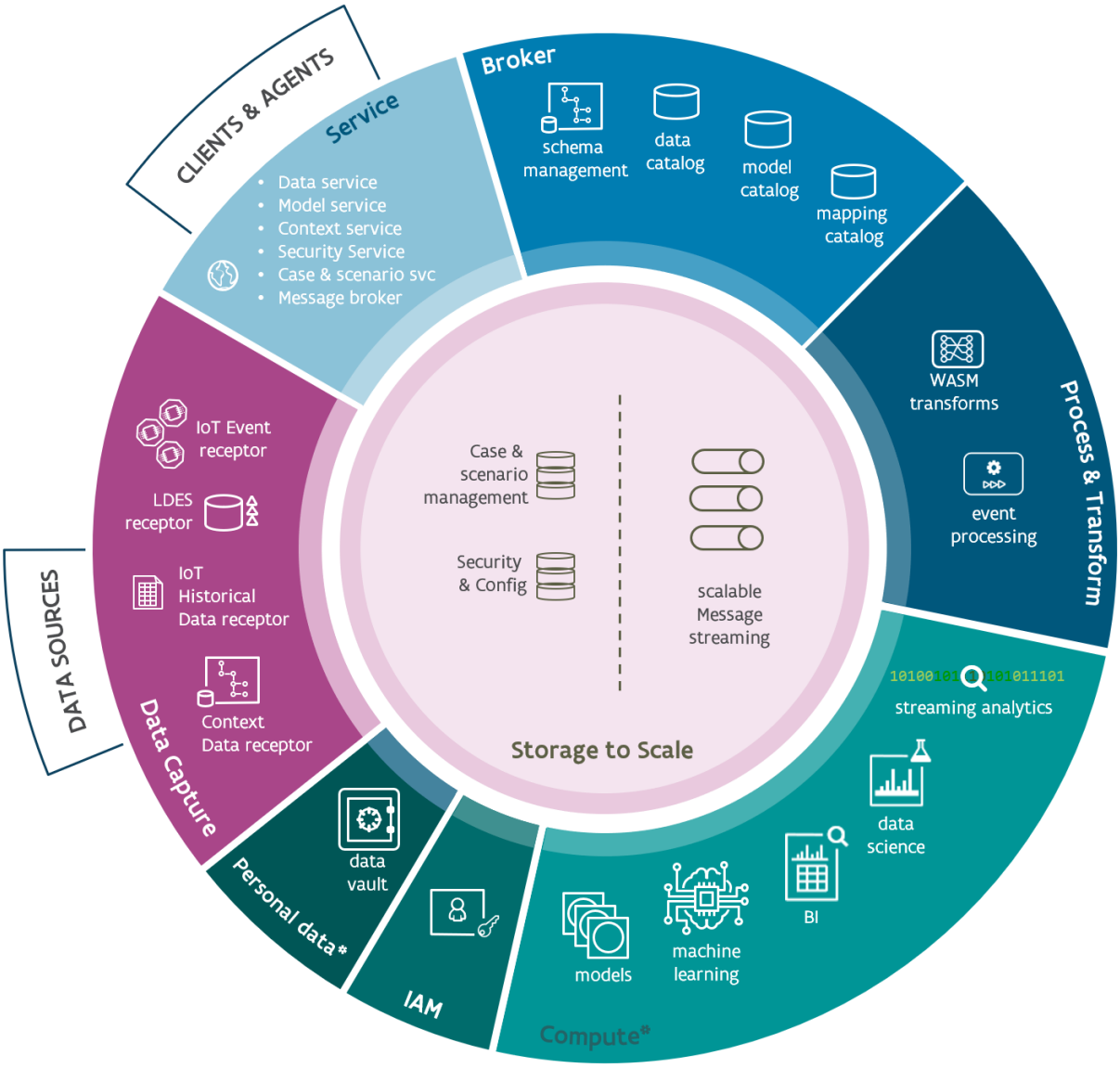

Figure 2: Conceptual architecture for Urban Digital Twins: The figure maps DT components onto the VLOCA conceptual architecture. This allows building a federated network of Digital Twin Infrastructure.

Capture

The capture domain is about connecting data. DUET provides several microservices for connecting data and making it available for other components:

- The IoT event connector for connecting to IoT event sources
- The IoT time series connector for connecting to IoT historical databases
- A REST connector for universally connecting to HTTP REST APIs
- An IoT context data connector which is essentially a more specialized form of the REST connector
- ...

Storage to Scale

When data is captured, it is transferred to the storage domain. Streaming is considered storage as well albeit usually in a volatile form. The DUET connectors will publish incoming data onto the central message streaming component via the DUET message broker service.

DUET does not store external data permanently. However, some data is kept inside the storage domain:

- user and access management data
- platform settings
- (meta)data related to assets (data sources, models, mappings)
- case and scenario data

Process & Transform

To bridge the gap between different standards or to enable reuse of proprietary data formats, DUET provides a mapping framework. Currently, the mapping can be done through an API but a configurable WASM based mapping plug-in is available as well. This will allow publishing default mappings between standards in a catalogue. Research on specifying mappings in function of the refined schema is ongoing and may provide the added advantage that only limited knowledge about coding is required to map between formats.

Compute

The compute domain groups all functionalities related to advanced data processing and reporting. DUET implements a Model Serving Pattern for processing data and making the result available again (see Section 5). This pattern is used for federating the use of models. This means that the connected models may be part of the smart data management infrastructure of an external partner. The broker component (see below) handles the connectivity between the different parties.

Computational models can be mobile as well. It is perfectly thinkable that a model, along with its connecting APIs and metadata can be shipped (e.g., inside a docker container) to a broker instance where data resides and run there. The result is shipped back afterwards or made available as a (remote) data source. This use case is currently not in the scope of the DUET project.

Service

The service domain makes internal data accessible for others. All of the DUET services (asset registry service, message broker, data services, model service, interaction service, management API, …) belong in the service domain.
Logically, external services also belong here. They are connected via the broker and the data receptors to the DUET core.

Broker

Discoverability and reusability of DUET assets to the outside world is handled in the Broker domain. Sharing data, models or mappings and managing metadata such as schemas, data formats, versioning, access, … is the responsibility of the DUET asset registry, that is composed of

- The DUET data catalogue
- The DUET model catalogue
- The DUET mapping catalogue
- Schema management & validation tools

Personal Data

Integrating personal data in Digital Twins requires special architectural considerations. We present a possible approach in Section 6.

Identity and Access Management

Section 6 also discusses security considerations for DUET. At present DUET uses Keycloak[7] to manage identity and access management. But to manage access at the level of cases, data sources and models, additional infrastructure will be needed. Special attention for sovereign identity management is needed given the context of federated access management.

2.3 Federated Asset Broker

An essential part of a Digital Twin Platform is the ability to share data sources, models and clients with the community. Typically, a digital twin brings together regional and local data sources with models from different providers and even reusable interface components. For instance, the implementation of a local digital twin cannot do without access to regional data (which may reside in regional digital twins) or data from neighbouring local authorities on traffic and pollution.

For the purpose of making data and other assets more findable and accessible, the smart data management paradigm foresees a federated asset broker architecture where the assets inside the brokers of the different platforms can be discovered by federating search queries across a network.

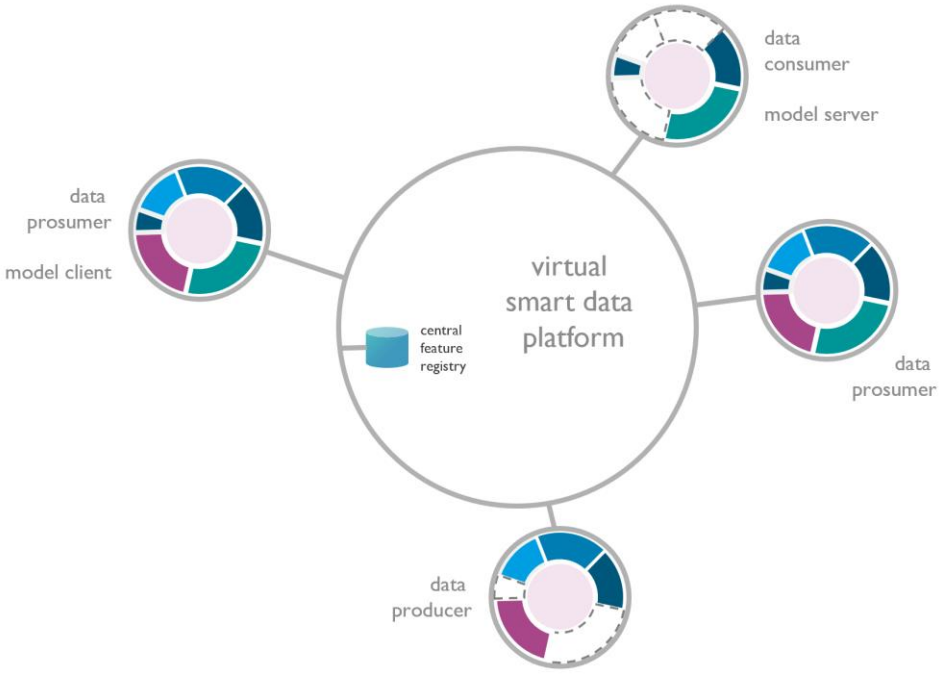

Figure 3: Federated asset broker: By interconnecting the brokers, multiple digital twins can form a federated network in which all assets can be shared as if they were residing in the local digital twin's broker..

[7] https://www.keycloak.org/ - Open Source Identity and Access Management

2.4 Onboarding IoT & Context Data

In deliverable 3.1, we discussed connecting IoT data through connectors, assuming that the data could be delivered by an IoT stack implementation such as a CEF context broker. However, in practice, data is often delivered in non-standardized ways. These ways range from CSV data dumps to pollable REST APIs offering arbitrarily structured data.

Aside from other ways of onboarding data, such as connecting NGSI-compliant IoT stacks, we look at another way to onboard (typically non-standard) IoT and context data, namely Linked Data Event Streams.

An introduction to Linked Data Event Streams (LDES)

The LDES specification [3] was created for dealing with such data sources and bringing them under governance[8]. The lightweight LDES approach can help capturing the data and publishing it as a scalable historical time series with a fairly simple specification.

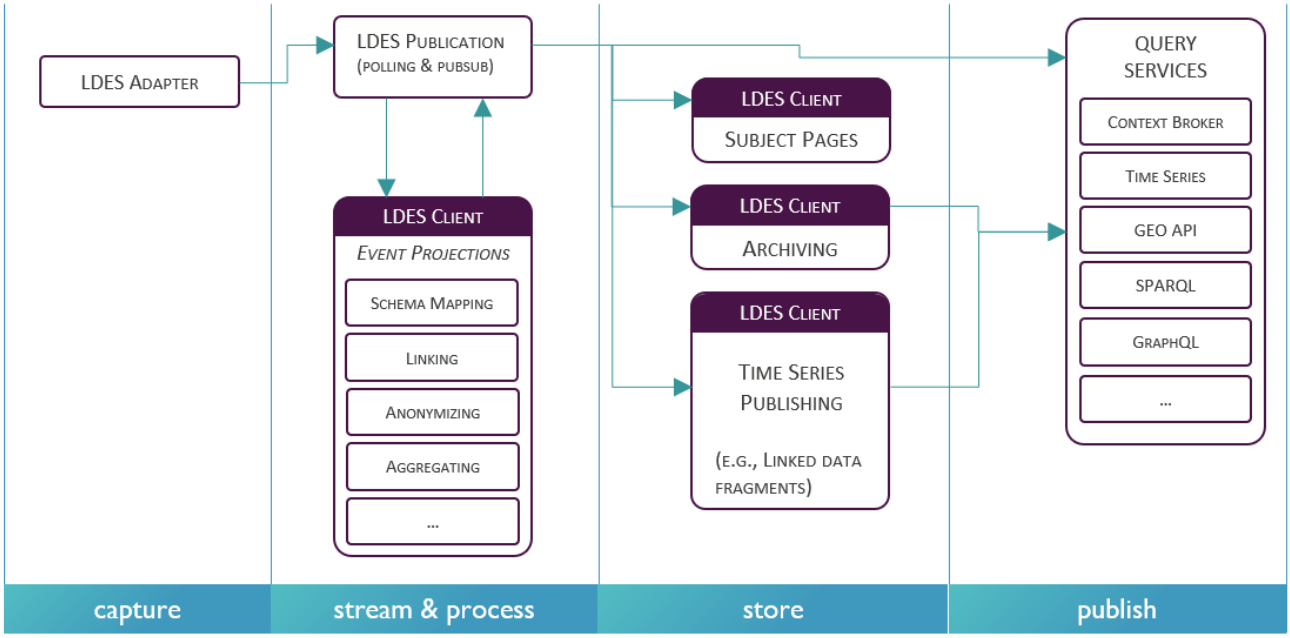

Figure 4: The LDES publication pipeline: By means of an adapter, the source system publishes events according to the Linked Data Event Stream (LDES) standard.

A Linked Data Event Stream provides a fragmented view of a complete dataset. The fragments (can) contain metadata to quickly navigate to the data you are looking for. Similarly to how hierarchical trees can be built for navigating spatial information (e.g., kd-trees), the use of the tree structure allows fast access, provided the tree is not too deep (a singly linked list is a special case of a tree). Since different users have different needs, multiple trees can be provided containing the same data. We can easily see that being able to drill down to a specific time can be essential for historical data sets. Similarly, for a geographical dataset a client may want to request only a smaller region of the whole. LDES provides means to add multiple fragmenters. This way different types of users can choose the optimal way to access the data. It is fair to compare materialized views with fragmentations. However each member of a tree must appear in all the other trees. This implies that

[8] See Asset Governance principle in Section 2.

aggregations are not fragmentations. That considered, fragmentations are more like full indexes, which obviously have downsides.

For datasets that grow on one side, like IoT time series data, one of the most straightforward ways of providing access is to publish a linked list of fragments, with the base of the tree being the most recent data. When the base of the tree is full, a new segment is created which can be cached for a long time, since the LDES components are immutable. Out of order arrival just means the data that arrived last is at the front of the list. A fragmentation that lists items by time of creation instead of time of arrival will be able to generate new blocks to accommodate for the out of order arrivals. An algorithm building a new fragmentation can cache previously viewed fragments, and when it encounters a fragment it has already seen, it can rely on its cache for the remaining data.

Clients of an LDES dataset start out by discovering the fragmentation that best suits their needs, and then walk the tree. The complexity of LDES can again be hidden behind a connector that is smart enough to untangle the tree and that can translate boundary parameters like a time range or a geographical region down to the most efficient ways to walk the tree.

Types of data

An IoT historical data set contains time-based events or measurements. Querying the data can happen in both the time dimension and the spatial dimension. It's also possible to provide a fragmentation based on sensor id, so that measurements for a specific sensor can quickly be found.

Similarly sensor event streams consist of events about the sensors: a change in location, calibration, and other contextual information. It's not hard to imagine a fragmentation ordered on sensor id first and in the time dimension second.

Base data sets are slow moving data sets. Obviously changes are possible, but the primary dimension in this kind of data is not time. Examples are: address to geolocation databases, 3d building data in various levels of detail, art collections, road networks, utility networks, … Although the primary use case for LDES is time based data, any data can be represented with useful fragmentations.

2.5 Data Lake House for IoT & context data

Note

Although the DUET project does not discuss data lake technology in much detail and considers it largely out of scope, we point out that the concept of replicating or storing data nearby algorithms, processing systems or reporting tools is very much compatible with the DUET architecture.

The role of data lakes and associated technologies is well established in big data processing. The DUET architecture allows components to be deployed on the same infrastructure as the data in order to limit latency. As such, Data Lakes are a technology that fits very well within the DUET architecture.

See Deliverable D4.5 for more details.

Terminology

We refer to data lake house technology as the collection of technologies that facilitate the collection of data sources for the purpose of (big) data analysis and reporting. From literature we find the following definitions:

- Data Warehouse: a repository of structured, filtered and pre-processed data. Data warehouses collect and keep data with a specific purpose in mind such as reporting.
- Data Lake: vast collections of raw data (structured and unstructured) grouped together in a logical architecture for which no specific purpose has to be determined. It is collected to do whatever analysis presents itself later on.
- Data Lakehouse[1]: A combination of the above technologies where on top of the data lake, structured data and indexes (data warehouse) are kept with a reference to the raw base data underneath (data lake data). This in fact allows to create a uniform access layer on top of heterogeneous data.

Raw Data in Data Lakes

Data collected by urban data platforms needs to be stored in data storage to be used for example for further historical analytics. This is the so-called cold path as opposed to the hot path being the streaming data. There is no one storage that can serve every purpose, therefore we need different kinds of storage for the use case or purpose at hand.

When data enters an urban data platform in its raw format, it is important to store these messages as early as possible, to prevent data loss. Storage for these items is often file or blob storage. and is often seen as multi-purpose cheap storage. No extra structure is added on top of this storage. It merely stores the data as-is and can be used for disaster recovery, replay or exception handling. If for example a certain conversion fails further down the line, due to an upgrade to the service. We can fix this conversion and replay the raw stream as if it would be played at that time, so no data is lost. Storage for this can be blob storage, file storage or s3 bucket style of solution.

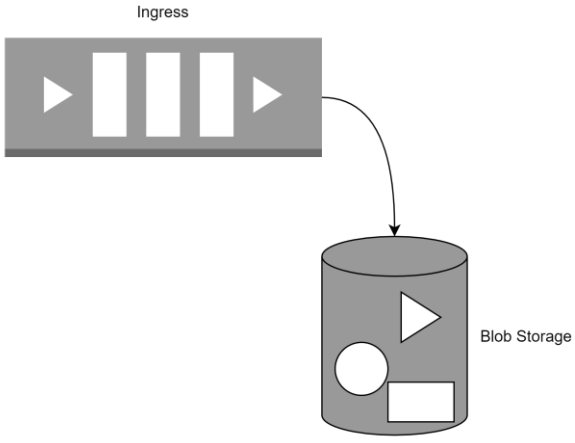

Figure 5: Raw data is ingested as-is in a first step.

Data Enrichment

Once the data goes further down the pipeline it becomes important to add context to be able to harmonize the stream to different data formats and standards. It can be important to have a high speed look up storage available for the needed context data. Examples are the location of certain devices of more administrative data like contact points and others. Linked Data Event Streams which are discussed earlier in this document can be a good candidate to act as a stream of such context data.

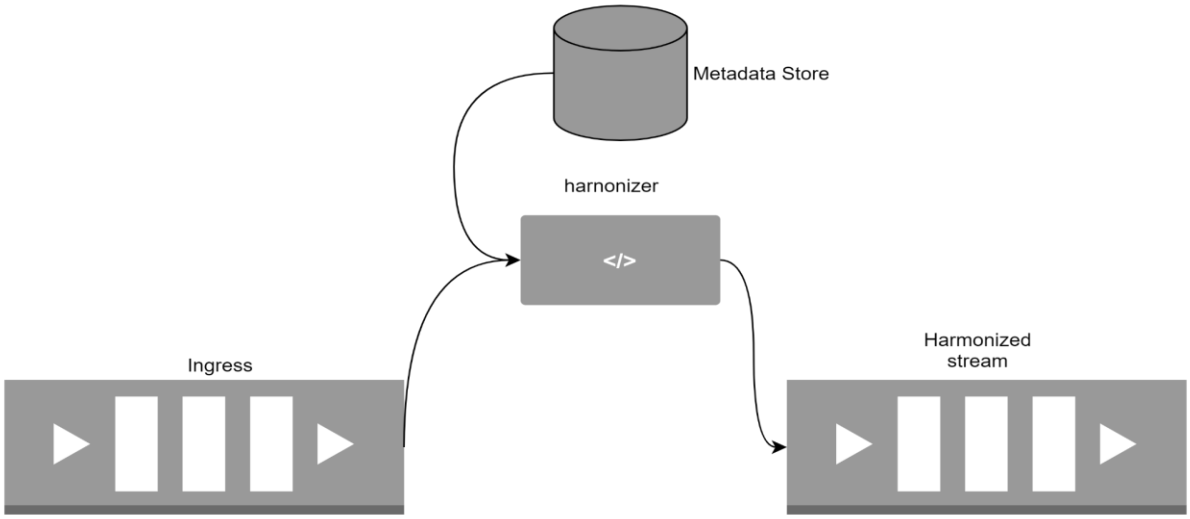

Figure 6: Raw data is enriched with context data from a meta-store.

The harmonized data needs to be stored in searchable and structured data storage. Examples of this are hierarchical data stores for example HDFS or Document stores like MongoDB. The purpose of this storage is to support the analytical jobs that will run on the data. Storage in hierarchical namespaces can enhance the speed of analytical jobs significantly.

The format in which the data is stored can differ across and in environments. The approach we want to suggest here is to store the data close to the original format, but as an underlying format to use json-ld. The advantage of storing the data in this underlying format is that we can then easily transform the data to the requested format of the algorithm.

Some types of algorithms require more speedy and specialized access towards data. Examples are algorithms based on time series. In these cases it can be that data needs to be stored in time series databases.

Data Lakes and Models

For performance reasons, models often need efficient access to data. This requires compact storage in local data-lakes where local means that the data can be accessed with very low latency. In practice this means that when models are connected to the DUET platform, they may want to keep a local copy of IoT data for their own purposes. An example of this is when a model needs to run analytics it will copy the necessary data to its local data lake (or cache), run the analytical process, store the result and then delete the copied data agan. The purpose of the local storage is here pure performance. In another case it could also be that the local storage first needs to transform the data for its processes and wants to handle this in batch. DUET supports this by allowing subscriptions to any data source as well as providing uniform access to historical data sources.

A similar reasoning can be made for the output of the models. When models produce great volumes of data, publishing that data as such may not be desirable. Instead, data can be stored locally and made available as a DUET data source through the general purpose receptors of DUET.

3 The IoT standards landscape

3.1 Existing standards

As in any other domain, several community initiatives have emerged in parallel for the IoT domain. Some of these initiatives have turned their insights into standards. The figure below provides a high-level overview of standards positioning them according to their main application area. On the left are standards that address mainly technical challenges: they define e.g. protocols for APIs. On the right are standards that capture the business information needs. As one can see on the axis, those standards are often technology neutral, but precise for a specific domain. E.g. sensor data for air and water quality.

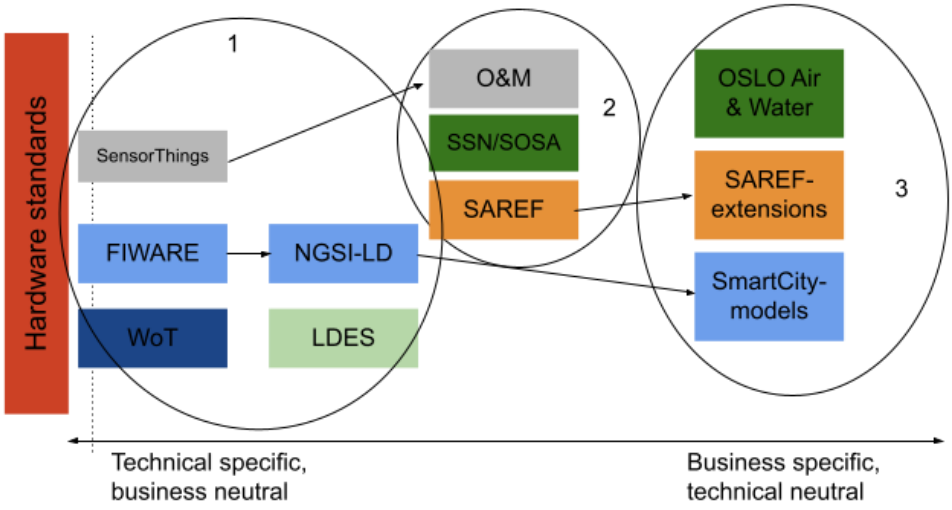

Figure 7: Categorizing IoT standards from more technical specific (left) to more business specific (right)

The groups of standards have been highlighted on the figure above:

1. 'API' standards → They define a protocol on how data can be retrieved
2. Generic meta models → They define top-level terminology such as entity, property, observations, measurement, etc…
3. Domain specific information models → They define terminology in the context of a specific application domain

An end-to-end application will apply standards of all 3 groups in a coordinated way. This creates ecosystems of integrated standards. In the figure ecosystems are shown by the arrows. Although one could consider combining standards across ecosystems, this usually comes with challenges to map the concepts.
Mapping standards in the group 2 (and consequently 3) is often possible to a large extent, however there might be unmappable situations such as where a term in one standard is partially covered by multiple terms in the other.
Mapping standards in group 1 is much harder: APIs are designed to work optimally in a chosen technical ecosystem, each also having their style preferences. Even if APIs are designed within the same technological context (e.g. REST with JSON payload), the API base technical behaviours might vary a lot. In application design the role of API style guides such as https://www.gcloud.belgium.be/rest/ is to provide a common ground for

these technical challenges. Unfortunately there is no universal style guide for API design, and since standards designing APIs must make decisions to create an executable setting, connecting systems based on different standards from the group 1 will require a substantial effort.
In addition to style differences there is often also a slightly different technical challenge addressed. Although there might be an overlap in the core challenge: receiving and submitting data from sensors, some are more natural to use for one context than another. E.g. one could be more adequate for wearables, others more for industry manufacturing.
All the mentioned standards address the digital layer. Hardware standards are not considered here. All standards displayed on the figure above will be briefly explained below.

OGC SensorThings API[9]

The SensorThings API provides an open-source and uniform API to connect IoT devices, data and applications on the Web. SensorThings provides two main functionalities, each of which has its own API: *sensing* and *tasking*.
The sensing part[10] provides a standard way to manage and retrieve observations and metadata from IoT sensors and provides similar functionality to the OGC Sensor Observation Service (SOS). The Tasking part provides a standard way for parameterizing - tasking - of taskable IoT devices, such as individual sensors and actuators. The Tasking part provides functions similar to the OGC Sensor Planning Service (SPS). The main difference between the SensorThings API and the OGC SOS and SPS is that the SensorThings API is designed specifically for the resource-constrained IoT devices and the Web developer community.
The Sensing Part of the SensorThings API was designed based on the model of Observations and Measurements (O&M). The image below shows how the Sensing API can be mapped to O&M.

SensorThings API Entities	O&M 2.0 Concepts
Thing (and Locations, HistoricalLocations)	-
Datastream	-
Sensor	Procedure
Observation	Observation
ObservedProperty	Observed Property
FeatureOfInterest	Feature-Of-Interest

Figure 8: Table describing SensorThings' data model can be mapped on the data model of Observation and Measurements (O&M)

[9] https://www.ogc.org/standards/sensorthings
[10] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

FIWARE[11]

An Open Source initiative defining a set of standards for context data management which facilitates the development of Smart Solutions for different domains such as Smart Cities, Smart Industry, and Smart Energy. FIWARE provides a framework of open source software platform components that can be combined with each other or with other third-party components. The core component is the FIWARE Context Broker, which enables the system to perform updates and access to the current state of the context information. The FIWARE Context Broker exposes an API, called FIWARE NGSI, that is used for integration of the components and by application to update or consume context information.

The current specification is called NGSIv2 and defines:

- A data model for context information, based on a simple information model of Entities, Attributes and Metadata;
- a context data interface for exchange of information through queries, subscription and update operations;
- an interface to query the availability of context information.

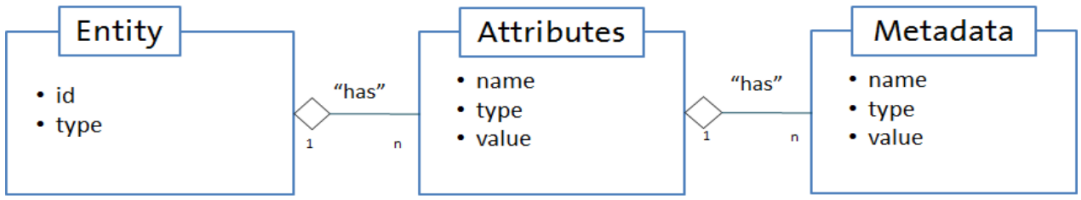

Figure 9: The main elements in the NGSI data model are context entities, attributes and metadata

The full specification is available at https://fiware.github.io/specifications/ngsiv2/stable/ . This specification is evolving to align with the NGSI-LD standard. NGSI-LD is a standard API for Context Information Management published as an ETSI specification. ETSI aims to bridge the gap between abstract standard and concrete implementations for use cases such as Smart Cities. With NGSI-LD the aim is to enable applications to discover, access, update and manage data and context information from many different sources. The image below shows a layered information model that is considered the NGSI-LD representation of real-world entities and their context information.

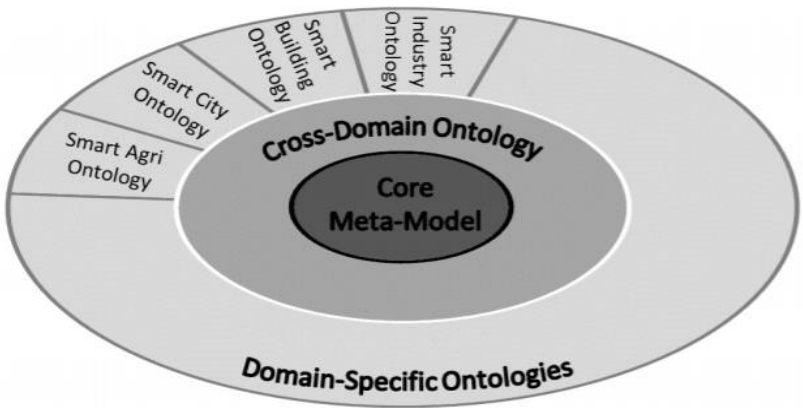

Figure 10: layered model to define the NGSI-LD information model

[11] https://www.fiware.org/about-us/

At the core, there is the Core Meta Model representing Entities, their Relationships, and their Properties with values. It contains the core terms needed to uniquely represent the key concepts of the NGSI-LD information model. The middle layer, Cross-Domain Ontology, provides commonly used constructs such as time and geographical location. As they are generally applicable to many domains, standardising their representation is valuable for cross-domain interoperability. The outer layer, known as Domain-Specific Ontologies or SmartCity models, can be created by extending the former two layers with terms from other ontologies. More information on the model can be found in the ETSI whitepaper about NGSI-LD.

➔ SmartCity models[12]

FIWARE is leading a joint collaboration program to support the adoption of a reference architecture and compatible data models. These 'Smart Data' models are grouped into subjects, each having their own git repository. The SmartCities domain contains information models to support the creation of a global digital single market of interoperable IoT enabled smart solutions. The FIWARE context broker, implementing the NGSI APIs, is the basis for breaking information silos in organizations. Publishing the data by using the common information models should increase interoperability.

Web of Things[13]

Web of Things (WoT) is a collection of standards developed by W3C to solve interoperability issues of diverse IoT platforms and applications. The majority of the WoT specification is about Things and Thing Descriptions. A Things is an abstract representation of a physical or virtual entity. A Thing Description includes the metadata and interfaces of a Thing in a standardized way, with the intention to make one Thing able to communicate with other Things.

The behaviour of a Things can be expressed by describing the interaction between the Consumer and the Thing, known as Interaction Affordances. An interaction affordance is a property of objects that can show to users what interactions are possible with the Thing. Things have four types of interaction affordances:

- Properties → Exposes the current state of the Thing (e.g. sensor value or status of the Thing)
- Actions → Allows to invoke a function of the Thing
- Events → Describes an event source that pushes data asynchronously from the Thing to the Consumer
- Web links → Hypermedia Controls to navigate to the Web

Web of Things is in many ways similar to FIWARE NGSI and work has already been done on how NGSI-LD can interwork with the Web of Things. The proposition is the create an NGSI-LD-WoT Adaptor that has a dual role in the architecture:

1. Offering applications an NGSI-LD interface to publish, consume and subscribe to information offered by Things, abstracted as NGSI-LD Entities.
2. Playing both the role of a client and a server in the WoT domain. For example, allowing applications to interact with Things associated with NGSI-LD Entities, and to consume functionalities provided by Things exposing them to/as NGSI-LD Entities. For instance, a WoT Property:Event can easily be exposed as an Attribute of an NGSI-LD Entity.

[12] https://github.com/smart-data-models/SmartCities
[13] https://www.w3.org/WoT/

More information about interworking NGSI-LD with the Web of Things can be found here.

Semantic Sensor Network Ontology[14]

The Semantic Sensor Network Ontology (SSN) can be used to describe sensors and their observations. SSN uses an vertical and horizontal modularization architecture, by including a lightweight core vocabulary, called SOSA (Sensor, Observation, Sample, Actuation) for its elementary classes and properties. As visible on the image, SSN can be mapped on Observations and Measurements by using an alignment module.

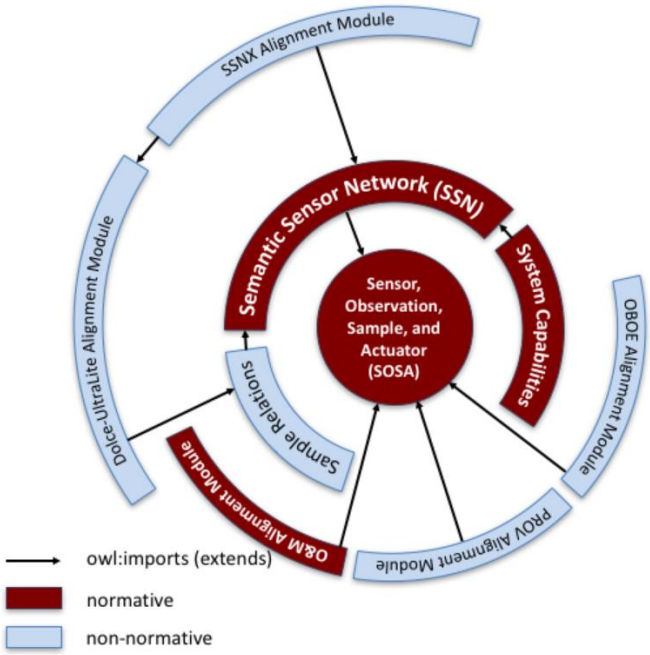

Figure 11: Image showing that the Semantic Sensor Network ontology includes SOSA as its core vocabulary and can be mapped on Observation and Measurements by using the O&M alignment module

Observation and Measurements[15]

Observation and Measurements (O&M) originates from work done by the Open Geospatial Consortium's Sensor Web Enablement (SWE) activity. Their goal is to establish interfaces and protocols that will enable a "Sensor Web", which will allow applications to access all types of sensors, and their observations, all of it over the Web. This standard defines schemas for observations, and for features involved in sampling when making observations. This results in models for the exchange of information describing observations and their results.

Smart Applications Reference[16]

The Smart Applications Reference (SAREF) is an ontology that facilitates the matching of existing assets in the smart applications domain. Their main goal is specifying recurring core concepts in the smart applications domain, the main relationships between these concepts, and axioms to constrain the usage of these concepts and relationships.

[14] https://www.w3.org/TR/vocab-ssn/
[15] https://www.ogc.org/standards/om
[16] https://saref.etsi.org/

The SAREF suite of ontologies provide definitions of generic classes that can be mapped to the NGSI-LD cross-domain ontology. Similar to NGSI-LD, SAREF also publishes domain-specific vocabularies in various domains, such as SAREF4CITY, a vocabulary for the Smart Cities[17].

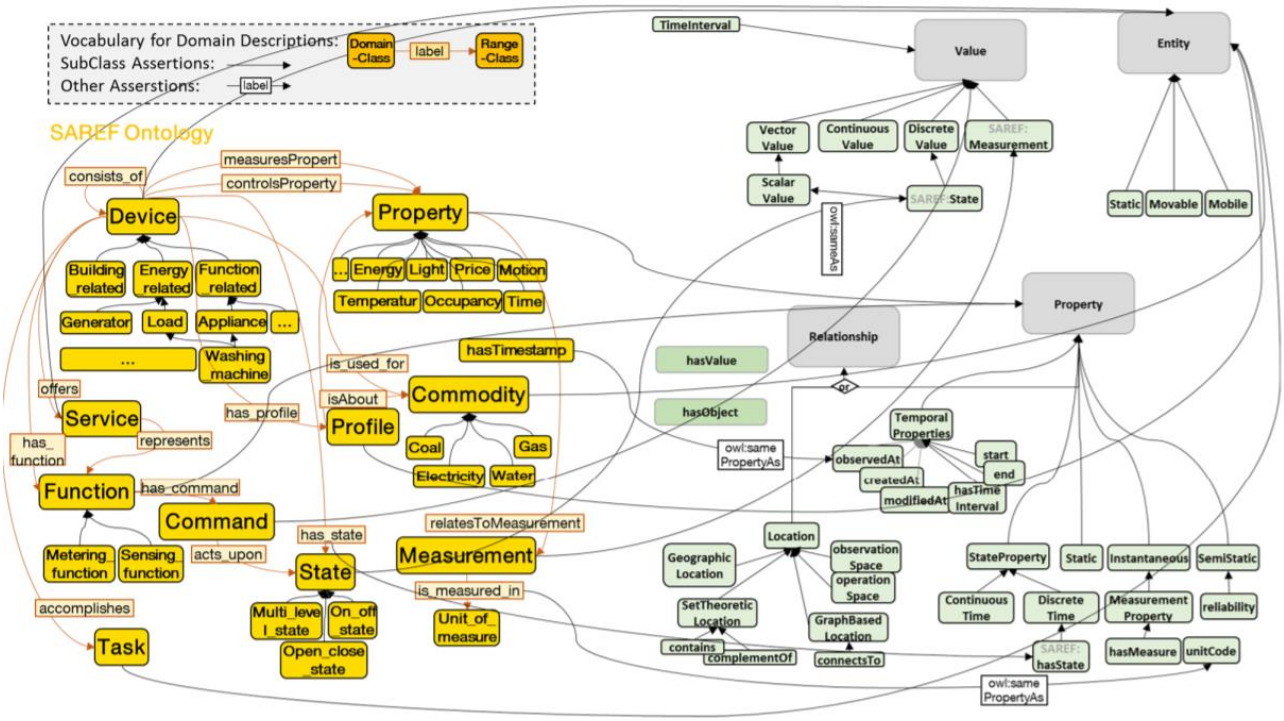

Figure 12: Mapping NGSI-LD to SAREF (Source: ETSI GS CIM 006)

Linked Data Event Stream

Linked Data Event Streams is a data publishing strategy that allows data publishers to do lifecycle management for objects in a dataset, and data consumers to synchronize with the history of a dataset, but also the latest changes. A Linked Data Event Stream is defined as a collection of immutable objects, such as an observation made by a sensor, or the version of an address. It can be considered an extension of a regular event stream, but it publishes interoperable data by re-using existing machine-readable data standards. More information on this specification can be found in deliverable D3.6 OSLO Extensions for the Digital Twin.
This part is documented in deliverable 3.6

OSLO Air and Water

OSLO Air & Water is a data model that is being designed according to the OSLO process and methodology and was initiated by the ODALA project. The model includes aspects from SSN/SOSA, O&M and FIWARE. More information about the realization can be found in deliverable D3.6 OSLO Extensions for the Digital Twin.

[17] https://saref.etsi.org/saref4city/v1.1.2/

3.2 Interoperability of IoT and Geospatial Data

Different geospatial urban data sources

Geospatial data used in urban data platforms come from different sources, formats and systems. Most of the existing urban geographic datasets are of 2D or 3D dimension, and formed by point, linear, polygon, or raster object types. Geospatial data can be stored in various formats, from the standard binary or text file-based formats to spatial relational databases. In order to integrate information from different systems sources, geospatial data is provided to the user or application as a service.

To ensure interoperability of different geospatial service data providers, there have been a progressive adoption at the National, European and Global level, of open geospatial web services standards, in particularly the ones approved by OGC, the Web Services standards (WMS, WFS, WCS, WPS, etc)

Integration of geospatial data web services and sensor web services

For easily integrating internet-connected sensors of sensor data services and location services OGC developed and proposed a family of open standards for sensor devices.

Sensor location is usually a key piece of sensor or sensor data information, and Sensor Web Enablement (SWE) standards make it easy to integrate this information into thousands of geospatial applications that implement the OGC's other standards.

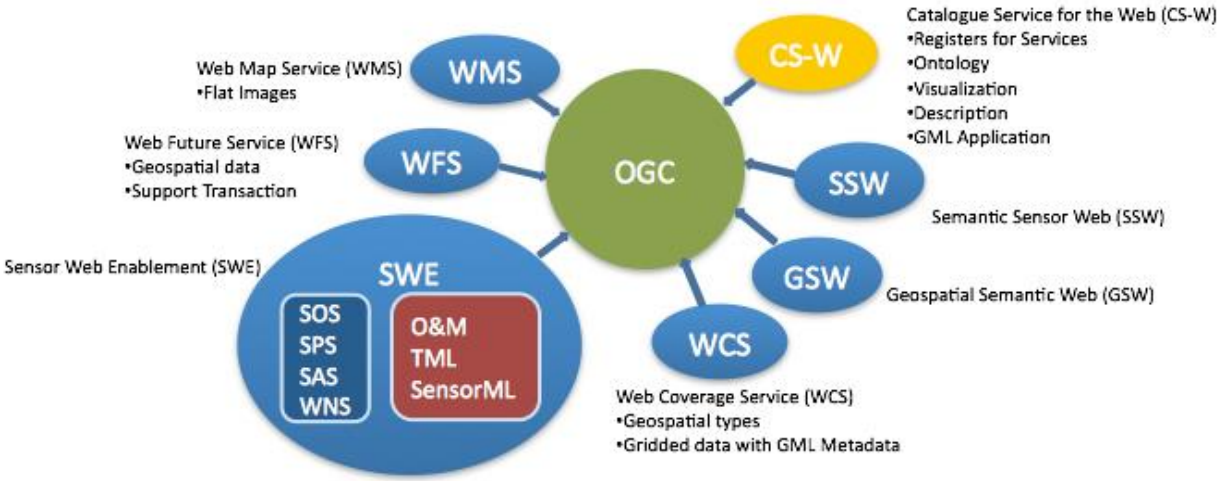

Figure 13: The OGC data standards landscape

Geospatial data from IoT devices

Data collected by IoT devices in urban data platforms generally include location information of the device and for some applications links to geospatial entities or objects.

The mentioned OGC SensorThings API[18] data model consists of two parts: the Sensing part and the Tasking part. The **Sensing part** provides functions similar to the OGC Sensor Observation Service (SOS) and the **Tasking part** will provide functions similar to the OGC Sensor Planning Service (SPS). The main difference between the

[18] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html#3

SensorThings API and the OGC SOS and SPS is that the SensorThings API is designed specifically for the resource-constrained IoT devices and the Web developer community. As a result, the SensorThings API is designed according to the REST principles and thus uses JSON as data format, and follows the OASIS OData protocol for querying and retrieving entities. Also an MQTT extension can be provided. The OGC SOS and SPS are built on the SOAP protocol and use XML als data format.

The Sensing part allows IoT devices and applications to CREATE, READ, UPDATE, and DELETE IoT data and metadata in a SensorThings service. The included geographical Locations of Things are useful in almost every application and as a result are included as well. For the Things whose location changed, the HistoricalLocations entities offer the history of the Thing's locations.

Note: New OGC API family of standards

Currently OGC is developing the new OGC Web API family of standards, built upon the legacy of those OGC Web Service standards, to define resource-centric APIs that take advantage of modern web development practices. These standards are being constructed as "building blocks" that can be used to assemble novel APIs for web access to geospatial content. (OGC APIs: Features, Common, Maps, Records, Processes, Coverages, Tiles, Styles EDR)

https://ogcapi.ogc.org/apiroadmap.html

Geospatially connecting data

Correlating two entirely disjoint data sets is not always a trivial task. Often these data sets lack shared identifiers and links that allow them to be correlated efficiently. Usually correlation can still be done using the geolocation but the use of different coordinate systems - of which there are plenty[19] - in data sets can complicate this as well.

In order to reliably correlate data sets, a data pre-processing step can be added for instance to harmonize the coordinate systems (to the extent possible) or adding the necessary links based on other properties.

Standards such as OGC Observations and Measurements allow to specify location details but these can take any form and shape. To accommodate a generic way of correlating data sets regardless of their scope, the addition of Point of Interest data may be considered. Adding references to points of interest allows to correlate the data sets more directly, reliably and efficiently. But POI data also creates new possibilities for digital twins. For instance, it allows for spatial auto-correlation across data sets[20]. The W3C point of interest core standard can provide a good starting point for a POI service.

[19] https://en.wikipedia.org/wiki/List_of_map_projections

[20] Spatial Autocorrelation: Close Objects Affecting Other Close Objects | by Anubhav Pattnaik | Towards Data Science

4 Virtualizing IoT data

A problem in digital twins is the fact that many disparate sources of data need to be unified somehow. Each of these sources defines its own internal data schema and these schemas might differ from each other. Terms that mean the same thing could be named differently across schemas. Or the value of one field could be the product of two fields in another schema. All of these differences cause incompatibility between data sources that require lots of custom code to validate and check each source as a consequence. Maintaining tailored integrations for each of these data sources separately is neither scalable nor feasible over time.

To transform between different data schemas in the current implementation, WASM (WebAssembly) transformers are used. A WASM transformer is a portable WebAssembly script that takes an incoming data payload (can be text, image, binary, etc.) and produces an output. These transformers can be written in Rust and then compiled to WebAssembly, but other languages that compile to WebAssembly are available. A json input and output schema are defined to specify the fields that are modified by the WASM transformer. This works well when only the technical schema is considered. However, a lot of the transformation details are hidden behind code and the transformer does not specify the domain constraints on the data in a clear manner. For that a refined data schema is needed.

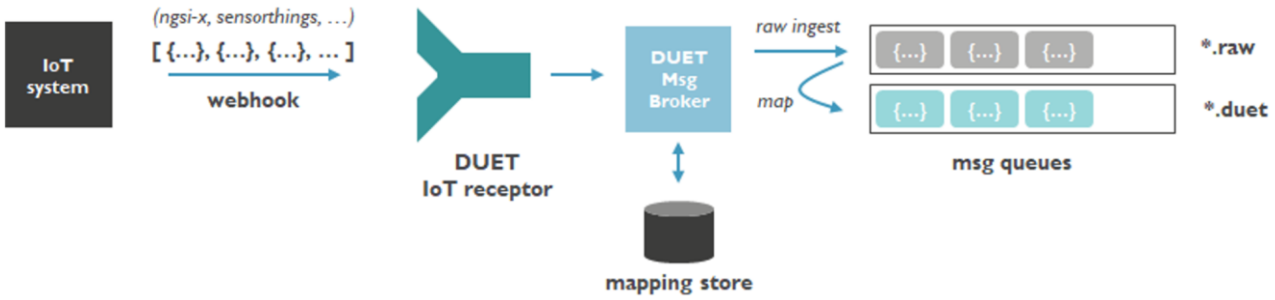

Figure 14: Virtualizing data sources through mapping in DUET: When data is streamed across the DUET message streaming platform, it can be mapped on the fly using (predefined) mappings. As such both the raw data stream and the mapped one are available to all users. Currently, WASM is implemented as a powerful mechanism. But maintaining mapping code can be complex.

4.1 Reference refined data schema

In order to tackle this problem, a reference refined data schema can be made. Refined data schema is explained in section 2. A reference refined data schema is a kind of 'master' schema, to which all incoming data sources have their own schema mapped to. This way, all data can be converted to conform to the reference data schema at ingestion time. This ensures data consistency and removes the risk of ingesting heterogeneous data sources.

A reference refined data schema contains domain specific constraints on each of the data fields. For example, if a data field is a percentage, a good constraint on this field could be formulated as follows: 0 <= field value <= 100. These constraints are written down using SHACL[21] (Shapes Constraint Language). SHACL is a validation language for RDF entities. SHACL defines 'shapes' which specify the conditions against which the RDF entities need to be validated. These conditions range from basic (e.g "Make sure the value of this field has minimum length 5"), to complex (e.g "Make sure that every instance of Hand has at most 5 value properties, that one of them is a Thumb and that 4 of them are a Finger'').

[21] W3C - Shapes Contraint Language - https://www.w3.org/TR/shacl/

An example SHACL shape might look like the following:

```
air_quality_2:Measurement-AirQualityIndex
       rdf:type          sh:PropertyShape ;
       sh:datatype       xsd:integer ;
       sh:minInclusive   0 ;
       sh:maxInclusive   10 ;
       sh:name           "Air Quality Index" ;
       sh:maxCount       1 ;
       sh:path           air_quality_2:AirQualityIndex ;
       sh:minCount       1 ;
       sh:description    "The Air Quality Index of the measurement" .
```

Figure 15: SHACL example of Air Quality Index

This shape specifies constraints on the AirQualityIndex property of a Measurement, such as the datatype (sh:datatype), the value range (sh:minInclusive and sh:maxInclusive) and whether the property is required or not (sh:minCount and sh:maxCount).

However, in order to do this, the mappings between refined data schemas need to be defined somehow. A possible risk here is that every data provider defines its own ad hoc mapping in code, increasing the complexity of the platform and a wild growth of different non-reusable mappings.

4.2 Mapping frameworks

A possible solution to this is to use RML. RML stands for RDF Mapping Language and is a generic mapping language that can be used to define custom mappings between RDF entities. Thus, RML can also be used to convert the technical schema of each data source to that of the reference refined data schema. RML can also be used to map SHACL shapes, to translate the domain-specific constraints.

Another option is to map SHACL using SHACL rules. SHACL rules allow for new SHACL shapes to be generated based on a set of conditions. An example of such a rule could be: 'If a Rectangle has a width property that equals its height property, then the Rectangle is a Square'. Which adds a property stating it is a Square to each instance that satisfies this rule.

These generic mappings generated through SHACL/RML can be saved and applied across data sources when possible, improving reusability and reducing the need for each data provider to define their own custom mapping.

A third option is to use ShExML (Shape Expressions Mapping Language). ShExML is a mapping language based on ShEx. ShEx is a schema language that describes and defines constraints on RDF data, using shape expressions. These shape expressions are similar to SHACL shapes, so in that sense, ShEx can be considered an alternative to SHACL. In most cases, ShEx can be translated to SHACL.

Using ShExML, it is possible to describe declarative mappings between data sources. The input consists of either JSON, XML, CSV or TSV and outputs RDF. ShExML can take multiple input sources and merge them. It

works by iterating over each entry in the input data, defining variables and performing manipulations on the data using these variables.

```
PREFIX : <http://example.com/>
SOURCE                                                                films_xml_file
<https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.xml>
SOURCE                                                               films_json_file
<https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.json>
ITERATOR film_xml <xpath: //film> {
   FIELD id <@id>
   FIELD name <name>
   FIELD year <year>
   FIELD country <country>
   FIELD directors <directors/director>
}
ITERATOR film_json <jsonpath: $.films[*]> {
   FIELD id <id>
   FIELD name <name>
   FIELD year <year>
   FIELD country <country>
   FIELD directors <director>
}
EXPRESSION films <films_xml_file.film_xml UNION films_json_file.film_json>

:Films :[films.id] {
   :name [films.name] ;
   :year [films.year] ;
   :country [films.country] ;
   :director [films.directors] ;
}
```

This example iterates over a json file and an xml file, both containing data on films. It merges both files and outputs the merged data as RDF.

The increased reusability and visibility of the mapping operations of declarative mappings are clear advantages over WASM transformers, which hide the mapping details behind code. However, more research is needed before a concrete technology choice can be made. A proof of concept that incorporates declarative mappings will bring more clarity on this choice.

4.3 Mapping in DUET

A WASM-based framework for mapping has been delivered in prior iterations. This allows developers to specify complex mappings from and to any kind of data format. This is very powerful and the WASM technology allows us to precompile, publish and reuse these mappings anywhere.

However, these mappings are not very transparent to users and lack a certain level of abstraction. Having a more abstract way of defining mapping enables users to create their own mappings, hence making the concept of an Urban Digital Twin Platform more open and accessible. In a next step we will consider existing mapping frameworks, assess their usability for this purpose and investigate improvements where necessary. Adoption in the context of DUET will depend on our progress in this matter.

5 Interactions, Models and IoT Context Data

5.1 Model serving

The goal of DUET is to create a digital twin platform where components can interconnect through generic interfaces. One of these interfaces is the model service which implements the model serving pattern. Model serving is a technique to connect models, data and clients in a generic way allowing models to be reused in different environments or for different purposes. Additionally, the approach can be used to interconnect models.

The model serving can be considered as an evolution of the OGC OpenMI [7] standard which specifies a way of achieving model interoperability at the software library level. The DUET model serving approach tries to apply some of its principles in a distributed service oriented architecture context.

The model serving pattern is discussed in more detail in Deliverables D3.5, D3.8 and also D4.3.

Model serving is achieved by

- Defining a data model for (simulation) model signatures
- Exposing a standard API for starting and stopping models and verifying the running status of a model
- Have the model use standard ways for consuming and publishing data[22]
- Publish models, their signature and other relevant metadata in an asset registry to make them discoverable and reusable
- Provide a way to store the result of model runs along with all relevant data (inputs, parameters, settings, version, …) to be able to analyse model performance and achieve transparency (cfr. fair and transparent AI [6])

DUET proposes an API and model signature through an Open API specification. It can be used as a starting point for the definition of a new OASC MIM (Minimal Interoperability Mechanism) that would allow simulation models and other algorithms to become more reusable across smart city applications.

5.2 Interactions & context serving

Cases and Scenarios

In order to understand how what-if scenarios are being supported we must first discuss how data is managed in this context. It is important that experiments can be run in isolation and that data from different experiments can be kept separate to avoid unwanted interference.

Within a case DUET distinguishes scenarios. An example of such a case is the implementation of a traffic circulation plan. Within a case, several scenarios correspond to alternative circulation plans that need to be compared. Adding data sources at the case level allows them to be shared across scenarios. Adding separate input and output data sources to a scenario instead of the case keeps these apart from the other scenarios. This way, experiments can be run in isolation.

[22] This is specifically relevant in the Digital Twin concept. Models can of course make use of 'local' data sources, exploiting efficient data access mechanisms for the purpose of performance.

There are two ways to have different inputs for scenarios, represented in the figure below:

1. Create a separate copy of the input data and publish/onboard it as a separate data source (green)
2. Create a difference layer on top of a shared base input data source per scenario (red and orange)

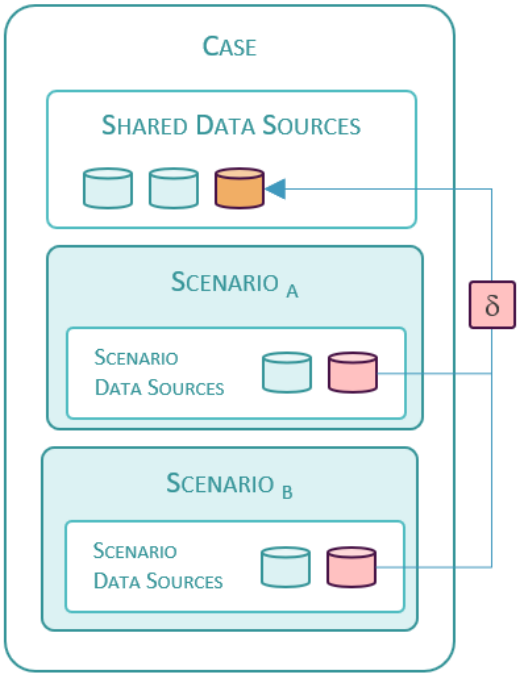

Figure 16: The organization of cases and scenarios in Digital Twins for supporting what-if scenarios.

City Context Data

City Context Data describes the (virtual) world we are looking at in an urban digital twin. This can be the road network on which traffic is moving or the buildings in the city. Digital twins allow us to interact with this world by adding, removing or changing elements in this virtual world and using simulation models to predict the impact of these changes.

Before discussing two major technological approaches to support these interactions, it is important to stress that when using the outcome of an experiment, it is always important that the exact circumstances of that experiment should be replicable for later validation. This is an important and at the same time impactful requirement in order for digital twins to be used as a tool for evidence based policies. Not only do we need to store the changes made to the environment for performing the experiment, but we should also be able to reconstruct the full historical context in which the experiment was created. This is a challenge given that cities change and evolve continuously. We refer to the time travel principles of Smart Data Management discussed in Section 2 which makes these requirements explicit.

Approach 1: a full copy

One way of interacting with the City Context is to create a copy of (a relevant part of) the actual state and use that instead of the original source of the input. Any changes we make are kept isolated from other data sets and we are free to change the data at will.

Changes can come from a client used by a city planner that is modelling a scenario. An example scenario used in the pilots is to allow a city planner to change aspects of an existing road network and use models to assess the impact on traffic, air quality and noise pollution.

But changes can come from sensors and other sources as well. The interaction service exposes an API that allows us to connect any event source to the hypothetical city context.

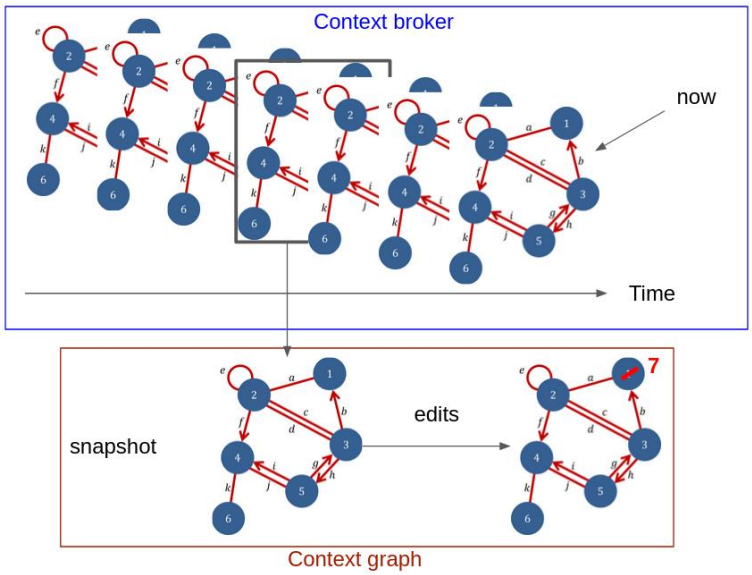

Figure 17: A conceptual picture of a context graph where hypothetical situations can be shaped and saved.

The resulting data set can be kept, stored and exposed as a DUET data source. Storing the data permanently will allow us to recreate the experiment at a later time.

Existing Context Broker Technology comes close to supporting all that is needed for implementing this approach:

- Keep networked entities and make them accessible using an API
- Allow changes to these entities
- Allow subscribing to these changes as a client

The downside of using a context broker is the property graph model that will require mapping infrastructure. This requires wrapping the context broker in a more client-friendly API.

What-if-scenarios can be implemented copying a relevant part of the city context data coming from a data source A into a separate context broker B, and making B available as a data source for visualizations and simulation models to use as an input source. Any changes can be detected by subscribing to changes in the context broker which in turn can be published as notification to the DUET message broker. This sketches the role of the context graph. Creating a snapshot of the state of a context graph, enables us to make changes without disturbing the original data source.

Approach 2: tracking changes

When copying the entire city context is undesirable, for instance because of resource limitations, an alternative approach can be to keep track of changes on top of the actual state. This approach can only be used provided that this actual state can be recreated at a later time (see time travel principles of Smart Data Management in section 2).

This approach is more difficult to implement since it requires the in-situ reconstruction of the changes made to the baseline. A technology that can be suitable to implement this approach is Linked Data Event Streams. With LDES, the underlying data structure is an immutable tree. Unfortunately the leaves of the tree point to the origin, so we can't reuse a leaf node in a fork of the event stream. However, we can lazily generate them: when a fragment is requested, retrieve the original data, replace the metadata to point to the changed LDES root, and update any entities that were changed. When a new change is made to the changed LDES, a new LDES root can dynamically be created.

Note: Implementing Interactions with LDES

With LDES, the underlying data structure is an immutable tree. Unfortunately the leaves of the tree point to the origin, so we can't reuse a leaf node in a fork of the event stream. However, we can lazily generate them: when a fragment is requested, retrieve the original data, replace the metadata to point to the changed LDES root, and update any entities that were changed. When a new change is made to the changed LDES, a new LDES root can dynamically be created.

Context Serving

Context serving is an architectural approach to making context city data available as a data source that allows the client to update it. Context serving in its simplest is shown in the figure below. Clients can update the city context data by using the interaction API that is exposed by the interaction service. The API updates the context database and notifies the digital twin of the updates. A model can use the new context to recalculate the outcome.

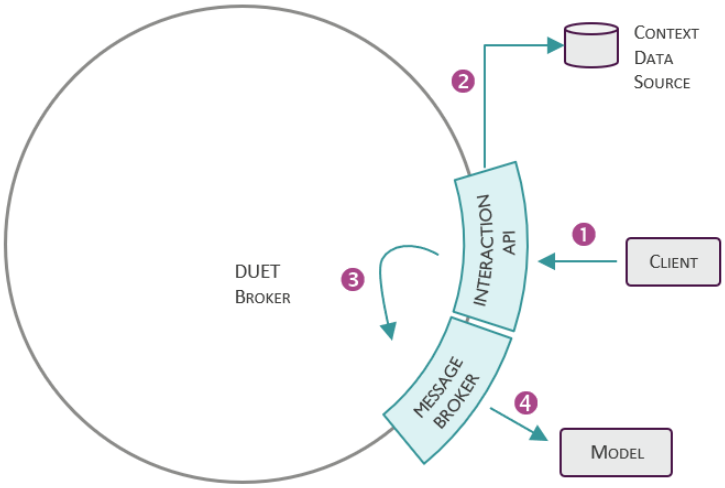

Figure 18: A basic context serving scenario

1 A client (or a model) sends an update to the interaction API via an HTTP REST web API.

2 The Interaction API processes this request by storing the change into a database that keeps the context.

3 The Interaction API pushes an update notification message onto a topic associated with the context data source.

4 The model subscribes to updates from the context data sources and responds accordingly.

5.3 What-If Scenario Data flow

A complete what-if scenario is typically more involved than the example below and often involves linking more models together. A well-known example is the entanglement of a traffic model and an air quality model. In the DUET Pilsen demo, we even add a noise pollution model to the mix. The overview below gives an idea of how changes to the context data can influence the outcome of the model. The example is also discussed in more detail in deliverable D4.3.

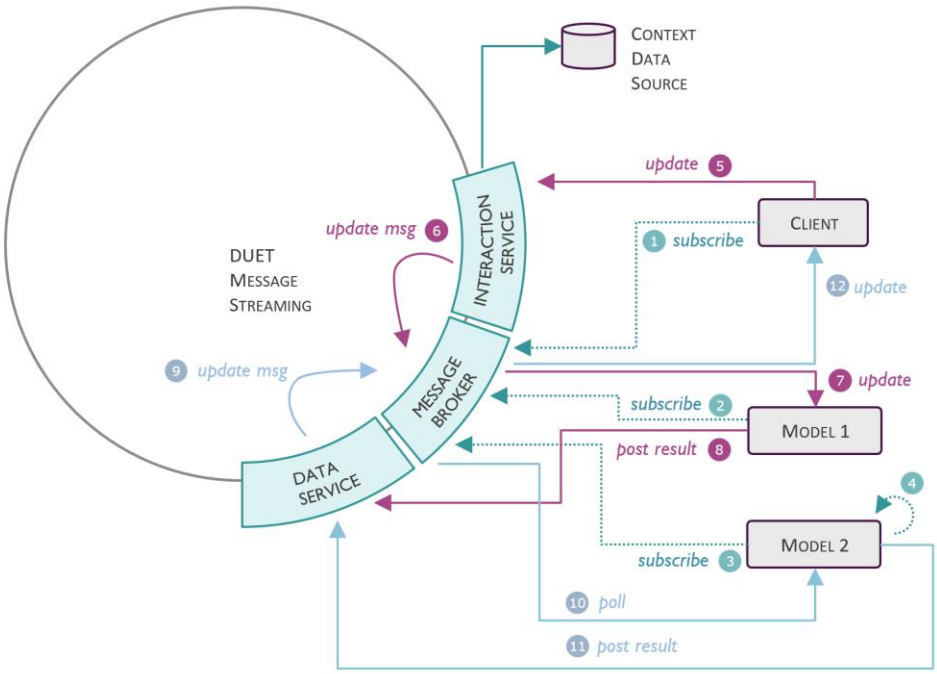

Figure 19: The what-if scenario data flow with model interactions

1 A **Client** subscribes with the message broker to update events of a **model result** data source associated with **Model 2**.

2 **Model 1** subscribes with the message broker to update events of a **context** data source.

3 **Model 2** subscribes with the message broker to update events of a model result data source associated with **Model 1**.

4 **Model 2** is started and waits for incoming results arriving at Model 1's result data source. Note: Model 1 can be started as well at this point, or it can be started when a notification comes in. This notification can be the result of an update in the context (through the interaction service) or simply the result of a button click.

5 **Client 1** sends an update to the I**nteraction Service.**

6 The **Interaction Service** publishes an update event associated with the update of **client 1** on the queue for the **context** data source.

7 M**odel 1** receives update messages from the **Message Broker** for the client update and responds appropriately - e.g., fetch the data needed for re-computation, trigger a new run, …

8 **Model 1** writes the result to the database attached as a DUET data source and notifies the data service that results are available.

9 When **model 1** publishes the results, a notification message is pushed on a queue associated with the result data source.

10 **Model 2** receives a result notification and fetches the data from the data service (not depicted). Model 2 absorbs these results in an ongoing model run.

11 **Model 2** publishes results to a data source. A notification message for new results is pushed onto a queue associated with the result data source.

12 The **Client** is notified of changes to the data as a result of publishing the results and responds appropriately. The client can respond to the result from Model 1 and 2 or only from Model 2. This depends on the client subscription(s).

6 Privacy & Security

6.1 IoT & Context Data Security

Security is an absolute necessity in IoT and smart data platforms since they play a more important role in the decision making and steering of organizations and governments. But they are often a gathering of complex systems and ecosystems which results in a complex structure to secure. New services that we introduce can lead to risks on the other systems and the complete risk assessments need to be conducted. We won't cover the complete set of security measurements, but we want to point to a security framework that was created by INESA. This framework contains a process to be performed when introducing a new or changed service. Important in this framework is that for each risk the organization needs to decide if they will take the risk, mitigate the risk or delegate the risk. An example of a list of risks can be found here: [Minimum Security Measures for Operators of Essentials Services — ENISA (europa.eu)]

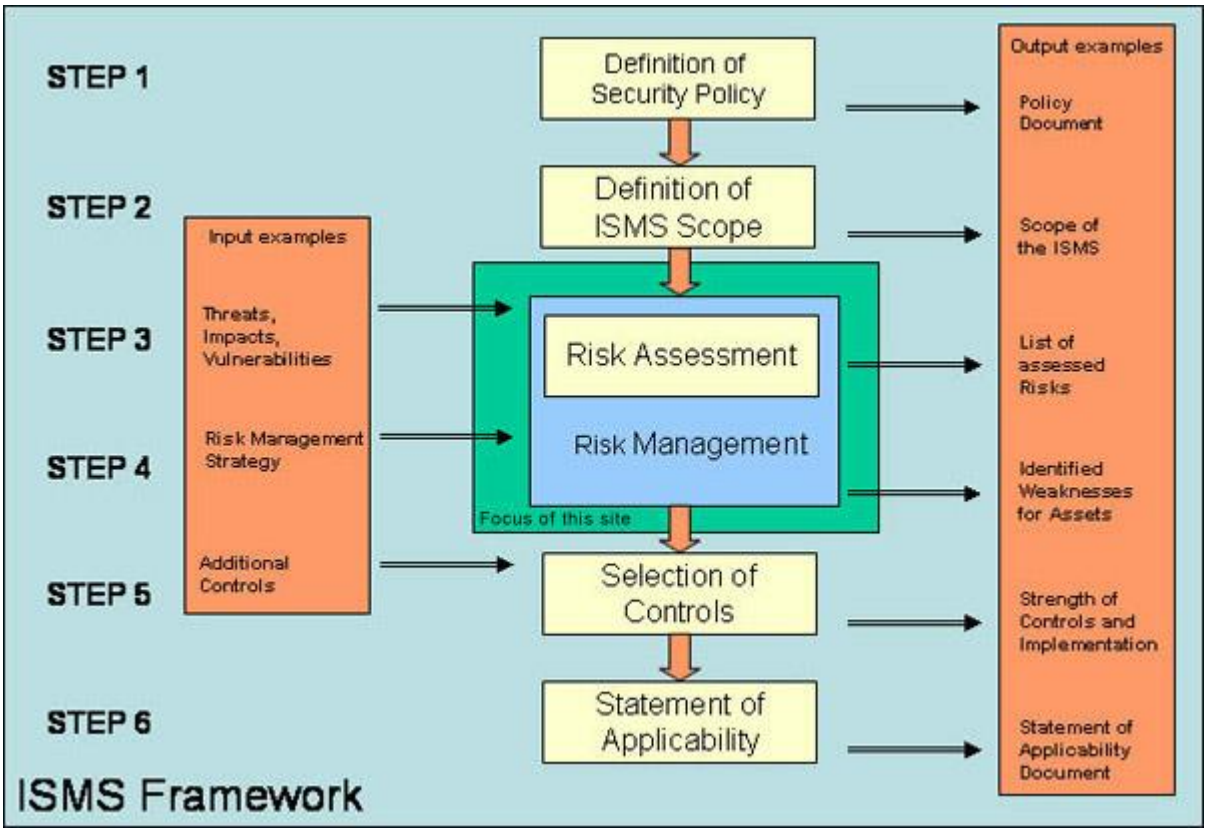

Figure 20: The ISMS framework from INESA

One important aspect we do want to touch upon is the protection of attributes. Since we collect the complete context of things with all its relations, it is important to have attribute based protection. Especially because we want to control the attribute access of the service to who wants to access the context. For example, service A can access att1, 2 and 5, but is not allowed to access the other attributes, while both service B and A can access att1 but B also all the others.

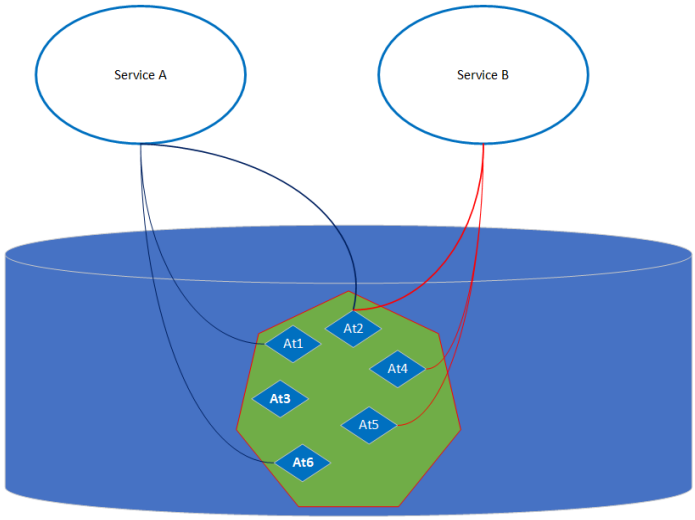

Figure 21: An attribute based protection example case

The way we want to tackle this is via web ACL on top of the URI. An example of this is the implementation of WEB ACL on top of solid: https://github.com/solid/web-access-control-spec

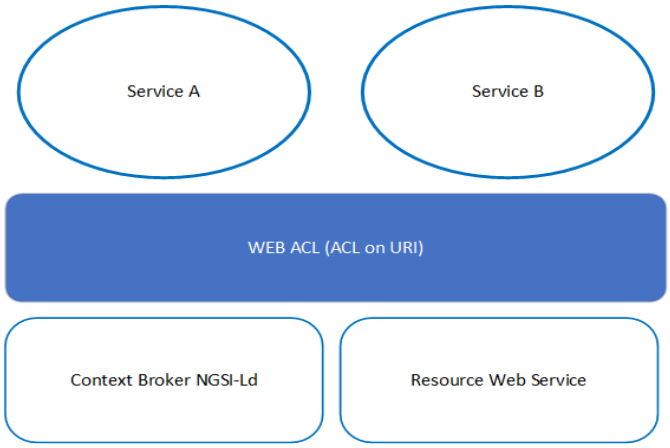

Figure 22: SOLID Web ACL layer for IoT data.

6.2 Personal IoT & Context Data Handling

In the scope of the POC there are no cases with personal data, but we foresee that digital twins will need to be able to handle personal data cases in the future. Therefore we want to discuss how we see this fit in the digital twin in this section.

For more personalized services it can be necessary to use personal data as a context for measurement data. For example, if we want to create more personalized mobility services we need to know the details of the person and need to handle payments and preferences. But since we need to follow the GDPR rules, the storage, collection and processing of personal data is very regulated. Next to that we want to give the citizen clear control and transparency of what is happening with his data and how it is stored. To be able to handle this we need to manage several services.

Identity management: the handling of the identity of the user and handling the keychain of identifiers for example, to let the users choose the type of identity they want to use like their social security number, but also their google account or services like itsme.

Permission management: Permission and consent of persons need to be managed. Persons can give consent, but can also consult and revoke consents.

Personal Data Storage: The storage of the data needs to be decentralized by default. Storage can be at the users' device. but can also be different governmental and private managed data pods.

Attribute access: services and users need access to attributes and security should be handled on attribute level.

In the diagram below we see an example of how the Vastuu group is setting up MyData.org principles in Helsinki. The city government governs the core attributes that citizens have in the different city services. With consent of the user these attributes and other attributes that users allow to be collected can be made available to other services. These services need to be registered in a service registry and undergo a vetting process before being allowed.
Users can use the identity of their choosing and these different identities are linked in a keychain of identities. allowing the user to pick the identity for the services at hand.
This model allows the users complete control over the data gathered, how and where the data is stored, which data a service can use and for what purpose and which identities can be used for the services.

To become truly interoperable, the identity management, service registry and data stores should be able to be linked to other services of other trusted parties like other cities or governments. Allowing a user to be able to use his identity and data where he/she chooses. This item will be further researched in MIM 4 Personal Data management.

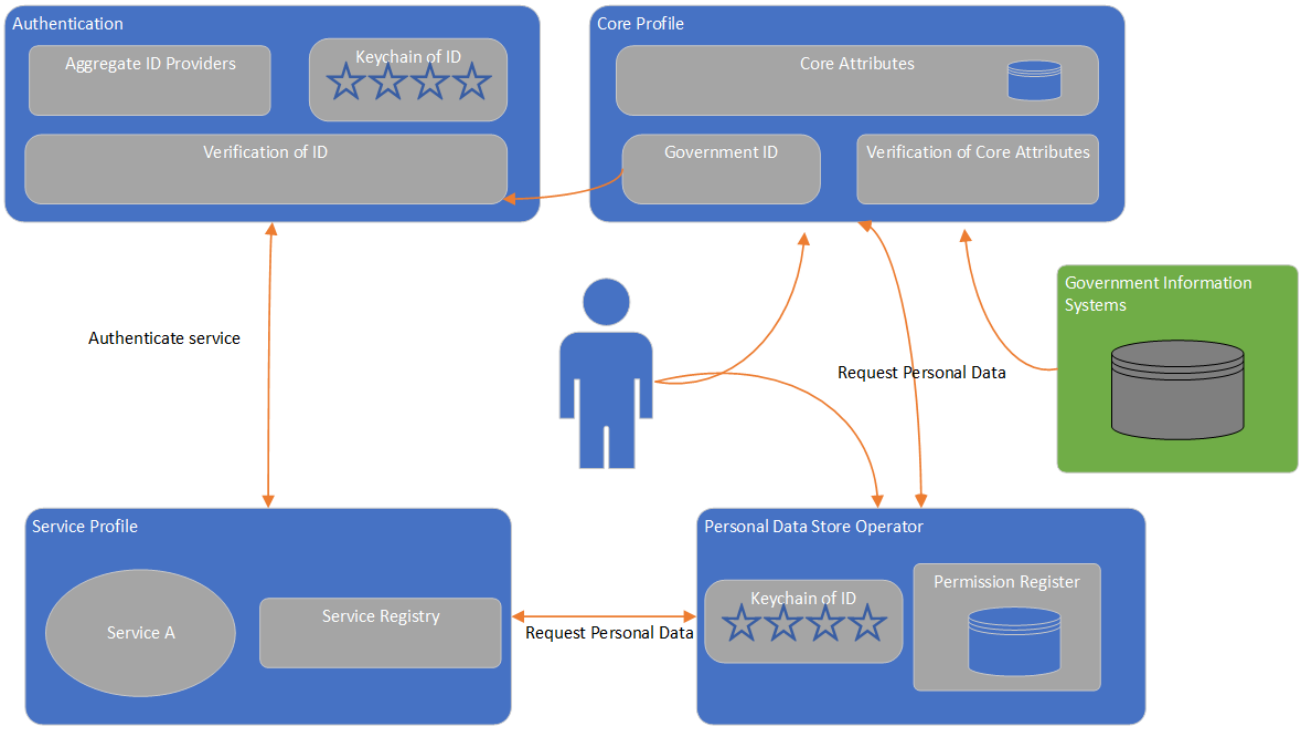

Figure 23: A personal data operator can act as a middle-man for exposing personal data securely and efficiently.

7 Conclusions & further work

In the Duet Digital City Twin alpha version, we presented a conceptual architecture built upon prior experiences with building digital twins. Those thoughts were brought into practice in the closed beta version. This allows the DUET consortium to learn what extra complexities still lurked beneath the surface.

At the end of the closed beta, we:

- Have implemented a prototype data catalogue that enables us to use data sources uniform across the architecture.
- Have the message broker, we have a running integration between the interactive frontend, providing scenario data over Kafka, to the different models. The traffic model reads this data and generates an output to the message broker, triggering the air pollution model. At the end of the chain, the front end again allows us to inspect the changes for both traffic and air pollution outcomes.
- designed the model catalogue, described how it is related to the model service and the model agent API
- designed how we can make interactions possible with existing data and how that ties in with the models

In the next phase of the project, we intend to:

- Add support for model serving by
- implementing and integrating the model catalogue, which will describe the specifics of a model as to inputs and outputs (model signature), so that other model vendors, currently unknown to the system, can integrate, get data as input, and generate output to be examined,
- having a model service implementation, which will use the model catalogue to know where the model lives and how to enable it,
- Having one or more model agent (reference) implementation(s) allows other model vendors to get started and integrated with the platform easily.
- Implement the context serving pattern with the interaction service, which hosts the context graph via the interaction API. This will allow us to create scenarios based on existing data by changing, deleting or adding items and using the result as input for the models.

Although we currently already have most of the required components, the focus was on proving that our design can work and learn how to proceed based on implementation challenges. The next phase will implement those designs and show future integrators how everything is tied together.

In the next steps, we will further scrutinise how the principles of smart data management align with the DUET architecture. This may affect some design choices and tools for the DUET implementation.

8 References

(1) Lorica, B., Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2020).
What is a Data Lake House?
Company Blog.
Retrieved May 1, 2021
from https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

(2) Agentschap Binnenlands Bestuur (ABB) (2020)
Vlaamse Open City Architectuur (Flemisch Open City Architecture)
Project site
Retrieved May 1, 2021
https://vloca.vlaanderen.be/

(3) Pieter Colpaert
Linked Data Event Streams
Living Standard, 1 April 2021
Retrieved May 4, 2021
https://semiceu.github.io/LinkedDataEventStreams/eventstreams.html

(4) D. Van Lancker, P. Colpaert, H. Delva, B. Van de Vyvere, J. R. Meléndez, R. Dedecker, P. Michiels, R. Buyle, A. De Craene, R. Verborgh (2020)
Publishing Base Registries as Linked Data Event Streams
International Conference on Web Engineering 2021

(5) T. Coenen, N. Walravens, J. Vannieuwenhuyze, S. Lefever, P. Michiels, B. Otjacques, G. Degreef (2021)
Open Urban Digital Twins – insights in the current state of play
Retrieved May 20, 2021
https://www.imeccityofthings.be/en/projecten/digital-twin

(6) M. Steen, TNO (2018)
Fair, Transparent and Trustworthy AI
Retrieved May 20, 2021
https://responsibledatainnovation.wordpress.com/2018/05/23/transparency-of-algorithms/

(7) Stanislav Vanecek, Roger Moore (2014)
OGC® Open Modelling Interface (OpenMI) Interface Standard
https://www.ogc.org/standards/openmi