



Deliverable

D3.9 Digital Twin data broker specification and tools v2

Project Acronym:	DUET	
Project title:	Digital Urban European Twins	
Grant Agreement No.	870697	
Website:	www.digitalurbantwins.eu	
Version:	1.0	
Date:	30/11/2021	
Responsible Partner:	imec	
Contributing Partners:	AIV ATC	
Reviewers:	Internal Thomas Adolphi (VCS) Hans Cornelissen (TNO) External Pieter Morlion Yannis Charalabidis Andrew Stott	
Dissemination Level:	Public	X
	Confidential – only consortium members and European Commission	

Revision History

Revision	Date	Author	Organization	Description
0.1	09.10.2020	Philippe Michiels	imec	Seeding
0.2	27.10.2021	Philippe Michiels	imec	first release for internal reviewers
0.3	29.10.2021	Sigve Veramndere	imec	adding TopBraid section
0.4	05.11.2021	Philippe Michiels	imec	Finish section on tools
0.5	11.11.2021	Koen Triangle	imec	Review
0.6	16.11.2021	Philippe Michiels	imec	Conclusion
0.7	17.11.2021	Philippe Michiels	imec	Relation to other documents, process first set of comments
0.8	19.11.2021	Pieter Morlion, Gert Vervaet, Andrew Scott	(external) AIV (external)	external review
0.9	28.11.2021	Philippe Michiels, Yannis Charalabidis	imec (external)	external review, process comments
1.0	28.11.2021	Philippe Michiels, Gert Vervaet	imec AIV	finalize document

Editorial notes

This is the **second version** of the specification. Although the architecture has evolved, it cannot be considered complete or final.

The term model has been used ambiguously throughout the project, its broad definition allows for this:

'a model is an informative representation of an object, person or system' ¹. In this deliverable we need to differentiate between **data models** and **simulation models** both of which are key elements of the DUET architecture.

A **simulation model** is a particular kind of mathematical model. It is conceived with the goal of emulating and understanding the behavior of a real-life system through software. Examples in the context of DUET are the air quality-, traffic- and noise emission models.

The term **data model** can be defined as an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities².

The reader may have to glean from context which of these is applicable when we refer to the term 'model', although we should use the terms data- and simulation model explicitly if necessary.

¹ <https://en.wikipedia.org/wiki/Model>

² https://en.wikipedia.org/wiki/Data_model

Table of Contents

Executive Summary	6
1. Introduction	7
1.1 Architecture Recap & Objectives	7
Recap	7
Objectives	8
Status of the implementation	8
1.2 High-level Changes	9
Message Broker (Replaces the app gateway)	9
Asset catalog (replaces data catalog and also covers model catalog)	10
Case Manager (Added)	11
Interaction service (Added - formerly known as Context Service)	11
Data connectors (Renaming of Data Source Gateway)	11
Model API (Removed)	11
1.3 Structure of this document	11
1.4 Scope and relation to other documents	12
2. Component Interactions	12
2.1 Cases, Scenarios and Interactions	12
2.2 Interaction Service	15
How the interaction service works	15
Example interaction dynamics	16
2.3 Example Scenario	17
2.4 Scenario Orchestration	19
STEP1: Setting up the case and the scenario	19
STEP 2: Starting (and stopping) a scenario	20
STEP 3a: Run experiments with batched updates	21
STEP 3b: Run experiments with streaming updates	22
3. Asset Catalog & Data Brokerage	23
3.1 Data Catalog	23
3.2 Model Catalog & model signature	26
Model metadata	26
Model calling	27
3.3 Brokering Ubiquitous Models	28
4. Tools	30
4.1 Data sources, models and the knowledge graph	30
4.2 Supporting model management	30
4.3 TopBraid	31
4.3.1 Data Assets	31
4.3.2 Business Glossary	31
4.3.3 Ontology	31
4.4 APIAPI	32

5. Conclusion	34
Current focus	34
Further work	35
Implementation: current status and commitments	35
6. References	36
Addendum 1: Extended interaction flow	37

Executive Summary

This document is an update of Deliverable D3.8 and reports on new insights in the realization of an Urban Digital Twin Platform.

The Closed Beta version of DUET which was released medio 2021 featured a 3D client with basic interactions support and several models working in composition to simulate the effects of changes made in the virtual city such as closing down a street. The implementation was a first step in showing that composing independent models for such purposes was indeed possible and was a first confirmation of the soundness of the architecture presented in D3.8.

In the Closed Beta version, the link between the client on one side and the models on the other was statically configured and their data was exchanged on predefined channels of the message streaming platform. This is not flexible as adding new models, clients and data sources would require intrinsic knowledge about the system, technical configuration in the back-end and tailor made coding on the side of the components.

The Open Beta release which is currently underway aims to make the connection between the components dynamic, allowing the user to combine data sources and models at will without the need for expert configuration or custom coding. These changes have come with new insights on the interactions of the different components that we discuss in more detail in this deliverable.

We have also studied new and ongoing initiatives in the area of data and model registration and publishing. This has helped us design a more crisp architecture for the data catalog component and brokerage functions in DUET. These insights are to be partly realized in the open beta release. The new design also specifies more clearly how discoverability can be achieved by including refined data schemas in the data source metadata.

1. Introduction

1.1 Architecture Recap & Objectives

Recap

The DUET T-Cell architecture is designed as a plug-in interface to support all these features. It exposes a message broker API that allows the components to connect to the T-Cell's internal *message streaming system* on which all data flows between the different components. Other APIs can be used to manage the registration of components (asset catalog) and the management of cases and scenarios (case manager) and also the management of the system and security.

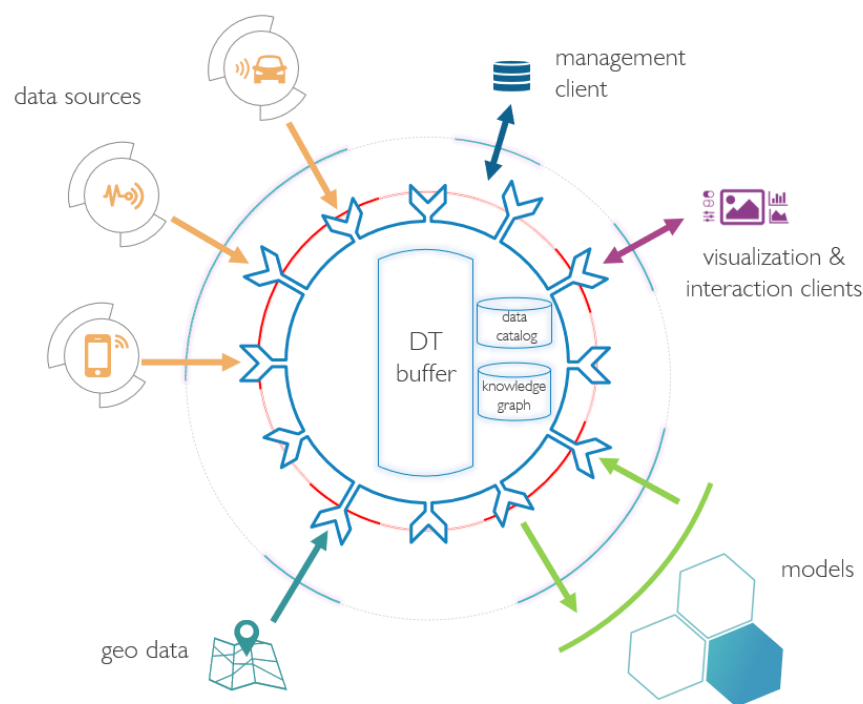


Figure: The DUET T-cell acts as a databroker connecting the data sources to the different DUET components.

The ultimate goal of the DUET architecture is to allow data sources, models and even digital twin client applications to be reusable and generic across different digital twins or in different cities.

Objectives

The objectives for DUET and this deliverable are two-fold:

Goal 1: Digital Twin Marketplace

From a business perspective, the idea is to create a platform as a set of components that facilitate the creation of a smart city data marketplace or even a digital twin Marketplace. This technical deliverable seeks to achieve this by making all of the essential components interchangeable and findable. This means that components such as interactive digital twin clients, dashboards, data sources, simulation models and other data processing components alike should all be able to connect to DUET using a set of interfaces on a shared platform to allow a decoupled architecture. This is a necessary precondition for a marketplace. Once this is in place, it becomes easy for suppliers to make their components available to be plugged in and as such become part of the Digital Twin Marketplace 0.1. We focus on the use of open linked data technologies to make integration of data and models easier.

Goal 2: Proof of concept implementation

Another objective is to bring this into practice and prove that the proposed architecture is valid by:

- Implementing the key components of the DUET core system in a first version:
 - Data and model catalog with discoverability (i.e., search functionality)
 - Case and scenario management
 - Support for real time and historical IoT data
 - 3D client
 - Interaction support
 - Generic data connection infrastructure
 - Generic model connection infrastructure
- Applying the technology on cross domain digital twin cases In Flanders, Pilzen and Athens using a variety of data sources and traffic, air quality and noise models

Status of the implementation

The current implementation already proves in a large part that the proposed architecture will work. Although the model catalog and the generic model connection infrastructure are still work in progress, they will be part of the upcoming Open Beta version.

1.2 High-level Changes

In order to further decouple the digital twin components, the architecture has evolved. Some components have been replaced by others and additional components were added to the architecture. The picture below shows a distilled drawing of the architecture highlighting the relevant components with colors.

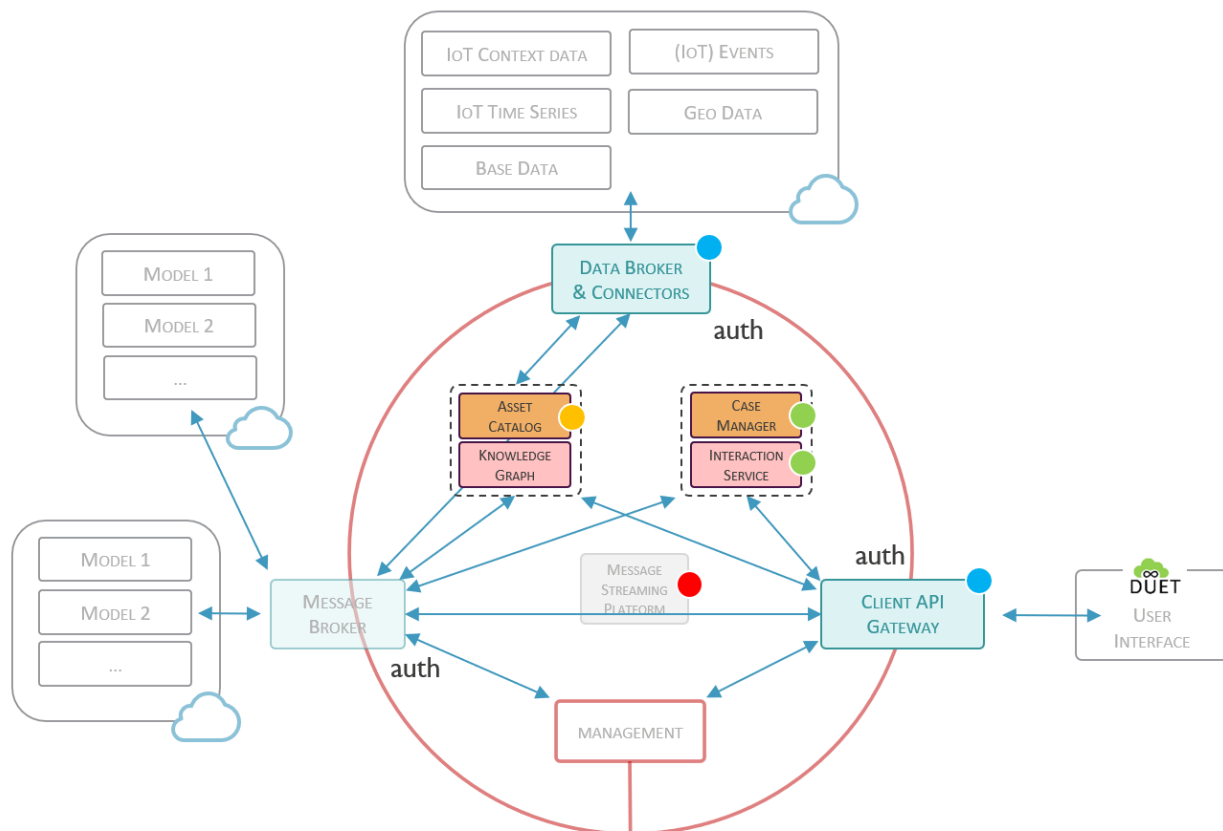


Figure: The distilled DUET architecture and its components. In contrast with the same picture in D3.8.

Components marked with a blue dot indicate a renaming of the component, those marked with amber indicate a replacement or extension, green dots indicate new components and a red dot indicates a removal. Note that the other broker components, such as the data connector also make use of the message broker to publish data/messages.

Message Broker (Replaces the app gateway)

The message broker component remains the way to

- (1) Publish data (by the data connectors) from an external data source onto the DUET broker so that clients and models can handle these in a responsible fashion,
- (2) Send messages to the DUET broker that are of interest to other components (models, 3D clients, dashboards), for instance for orchestration purposes,
- (3) Subscribe as an external component (such as a model or a 3D client) to messages of interest pushed by other components. These can be data events or orchestration messages.

Other requirements are that:

- The message broker allows you to subscribe to one channel for all orchestration messages (messages not related to data, such as a model signaling the completion of a run or the user interface sending a request to start a model run). Preferably, the messages can be filtered on a scenario ID (see below in this document) if it is specified.
- The message broker allows you to subscribe to data coming from a datasource by using the data source id as provided by the data catalog. More details on how data formats can be mapped are discussed in D3.8.
-
- The message broker can support multiple protocols for the consumers of the messages (i.e., the external components). The basic REST polling API can be extended with web sockets for instance to enable more responsive behavior of 3D visualization clients.

Asset catalog (replaces data catalog and also covers model catalog)

The role of the data broker was elaborately discussed in Deliverable D3.8, the previous version of this document. We refer to this document for the interested reader. Next to the data catalog, the model catalog is being introduced in the first version. We will refer to the combination of the model catalog and the data catalog as the asset catalog.

In Deliverable D3.5 and D3.8 we introduce the model (agent) API as a way to explicitly start or call a model. The API specifies which data sources are consumed by the model, what data sources contain the model's output and lastly the configuration in which the model should be run.

This was the basis for the definition of model signatures. A model signature describes a model in terms of its expected input data and output data parameters including their formats as well as the required configuration parameters required to run the model. All of the data sources and parameters should be described in detail in the metadata in the model catalog.

A first formal information model for describing model metadata is given below. It still needs formalization and can be extended as seen fit by model publishers. A model metadata information model is proposed in [3.2 Model Catalog & model signature](#).

Note: Model Signature versus Model Call

It is important to understand the difference between a model signature and a model call.

A model signature describes a model in terms of what kind of input data it expects (data schema) and how it expects it to be delivered (technical interface). The same goes for the output data it produces. It does **not** point to specific data sources.

A model call is an application of a model in a specific context (a scenario in the case of DUET). Here references to actual data sources for both input and output parameters are being specified.

Case Manager (Added)

Data publishers and model providers can register their assets in the catalog for city administrators to apply them in decision support cases. The cases are created by these administrators by means of a client interface but they are stored in the digital twin case manager.

The case manager allows city administrators to create smart city cases such as a comparative study for circulation plans in a city. The case manager allows to fully document the case by providing a high level explanation, supporting document etc. Typically a case will have multiple scenarios below it, for instance to compare several alternative circulation plans and how they perform in terms of specific KPIs.

The concept of a scenario allows experimentation with data and settings in isolation from the other scenarios. By selecting different output data sources for the models per scenario, the results can also be kept separate for comparison. This way, a model run in one scenario can never overwrite the results of the other provided the data sources are not shared across scenarios. This is a simple but effective way for achieving this kind of separation.

Interaction service (Added - formerly known as Context Service)

In this version of the architecture the context graph has evolved into the interaction service. The interaction service is a more proper name as the service mainly serves to keep track of the changes made - either by a user in the digital twin client, or by models that have side effects on the status quo, such as recomputed traffic intensities for certain roads.

The interaction service does three things: (1) it keeps track of all the changes made so that the new situation can be reconstructed from the baseline scenario at any time by any one component, (2) it publishes all changes on the Message broker of the DUET core system so that other components can detect the changes and act accordingly (3) it exposes an API that allows to fetch changes made after some point in time so that changes can be processed in batch.

Data connectors (Renaming of Data Source Gateway)

The data connectors have remained the same. They allow connecting data sources to the messaging system exposed by the Message Broker. They have been discussed in full detail in D3.8 and we refer to that deliverable for a full description.

Model API (Removed)

The model API has been removed from the architecture because all of its responsibilities can be absorbed by the other components such as the Message Broker and the Model Agents. Its core function was the ability to control models from the API. This has been eliminated in favour of publishing messages to the message broker that can be subscribed to by the Model Agent.

1.3 Structure of this document

The remainder of this document is structured as follows:

- **Chapter 2** discusses how components interact with each other and explains how orchestration is achieved in the current system. It also discusses how this could be improved upon.
- **Chapter 3** discusses how the data catalog and the model catalog relate and explain how scenario management brings models and data together for decision support.

- In **Chapter 4** we give an update on the tools that we use and also how other tools can be integrated, notably towards linking data sources with the central knowledge graph.
- We conclude in **Chapter 5**.

1.4 Scope and relation to other documents

This document amends Deliverable D3.8 with new insights in the realization of an Urban Digital Twin Platform. Most of what is discussed here builds upon the previous version of this document and it is recommended to read this document first.

The discussion here focuses on the best way to realize the proposed digital twin of independent / decoupled components. For more details on the components themselves and cross-cutting concerns such as security, we refer to other documents:

- **D3.2 IoT stack and API specifications 2** describes the components onboarding of data into DUET (updated version of D3.1)
- **D3.4 Smart City domains, models and interaction frameworks v2** describes the models that the different partners can provide to DUET (updated version of D3.3)
- **D3.5 Cloud Design for Model Calibration and Simulation** discusses how models can be run in the cloud and connected to DUET, using the potential of HPC infrastructure
- **D3.6 OSLO Extensions for the Digital Twin - current status** describes a standardized process for developing smart city standards for use in digital twins. As extensively discussed in prior deliverables and repeated in this deliverable, having a common vocabulary will help a great deal in building the digital twin which mandates a framework such as OSLO.

This deliverable also does not discuss an overall architecture and implementation or deployment plan. This is discussed in **D5.1 System Architecture & Implementation Plan**.

For a detailed discussion of security matters, we refer to **D3.11 Multi Layered security model specification** (updated version of D3.10).

2. Component Interactions

2.1 Cases, Scenarios and Interactions

The general idea of DUET as a connector system for digital twin building blocks requires that there is a place where all the building blocks come together. This is the responsibility of the case management service where cases and associated scenarios can be managed. The case manager is part of the core and exposes an API for any client to manage.

Case

A case contains the full details of the problem that is being tackled. It can be about implementing a circulation plan, examining crowding effects on planned events, the traffic impact of a new housing development, etc. It is a full circumstantial description of all the factors involved with the purpose of explaining the matter to all stakeholders, including citizens.

When a conclusion has been reached, it may also report why the decision was taken and based on what data and insights.

Scenario

Within a case, different scenarios may be explored. The scenario level contains everything needed to do the necessary comparisons. Scenarios within a case can be used to assess alternative solutions for the problem by means of data and/or simulation models that examine the impact of each solution. Scenarios are expressed in terms of what models are applied to what data sources. This is static configuration that is stored under a scenario record.

Alternatively, scenarios can also be used to evaluate the performance and accuracy of models. This can be anything like comparing different configurations of the same model or comparing different models all together.

Each scenario references the data catalog for data source of interest. These can be input data sources for models or visualisation. But also output data sources can be added. The fact that output data sources are kept apart in the scenarios allows comparison of the results later on.

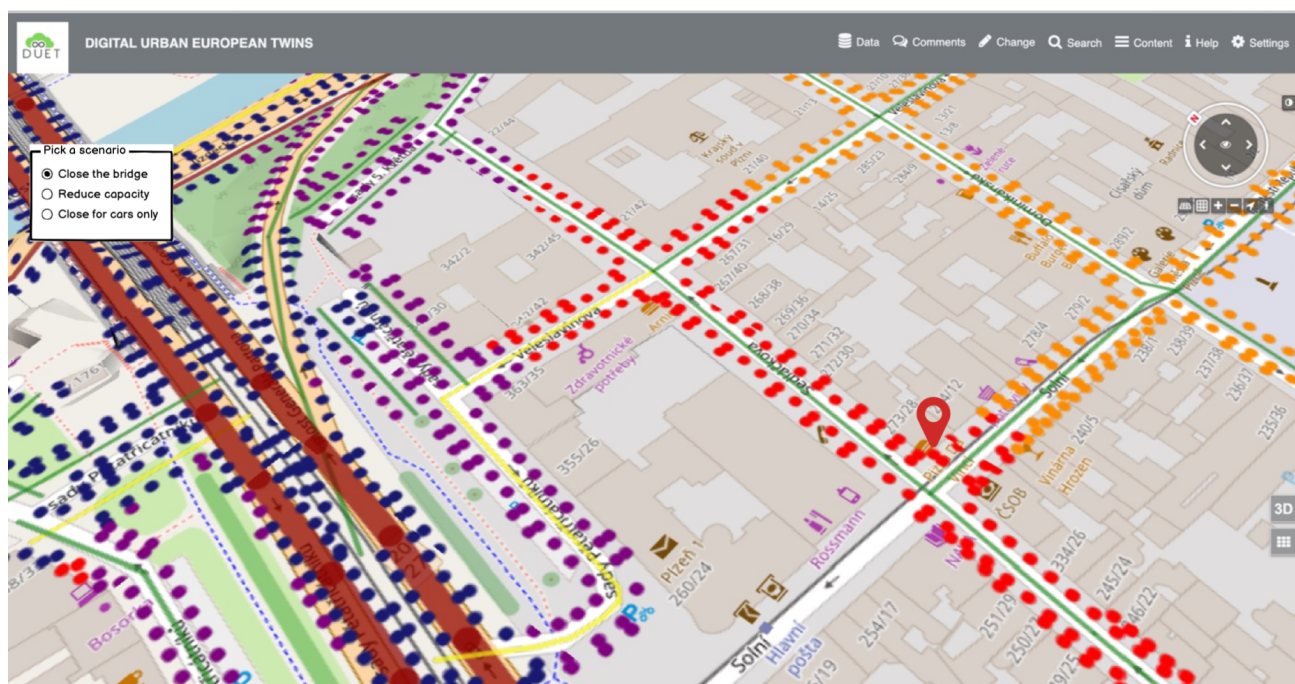


Figure: Mock for UI to allow citizen to switch between different scenarios of a case

Data Source Reference

Models do not contain data sources themselves, they merely reference the data source in the data catalog using a data source identifier. The user can explore the data catalog and add any data source deemed relevant to the scenario. See the picture below and the caption to make this more clear.

Model calls

The scenarios do not contain actual models, the models are added by referring to them using an identifier from the model catalog. In the scenario it is specified which input, output and configuration is used for that model within this scenario.

We refer to this principle as model calling, since it resembles calling a function with certain parameters.

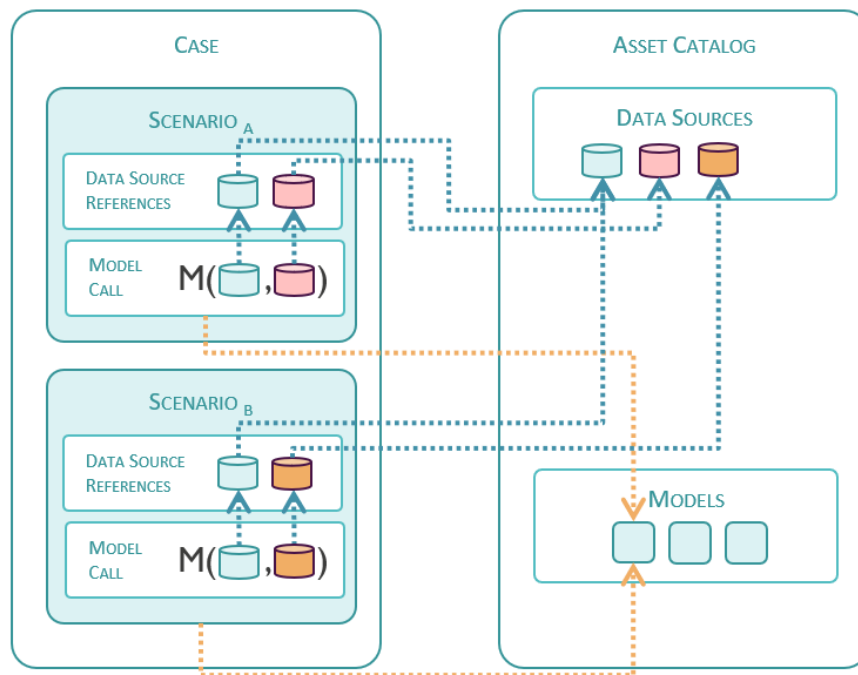


Figure: Within a case, a scenario contains references to model(s) and, separately, references to data source(s) that **in this scenario** would be fed into the model.

Note:

As of today there is no semantic description of cases and scenarios, nor is there a schema or standard API for its management. As digital twin technology matures, such a standardization may foster further interoperability between digital twin building blocks.

2.2 Interaction Service

In the previous version of this deliverable (D3.8), the context graph was introduced as the component that facilitated interactions in the Digital Twin. In order to better capture the true role of the component we have renamed it into the Interaction Service.

The role of the interaction service is to keep track of changes that are taking place in the hypothetical context of a scenario. These changes can come from

- a city planner, making changes in the virtual representation of the city such as closing down roads, adding sound barriers, implementing circulation plans, etc. In DUET, closing of a street is implemented, other types of changes are here as examples for future work.
- a model that updates the state of the virtual city as the result of a computation such as new traffic intensity, different water levels in sewage systems, updated air quality values, etc.

Note:

The interaction service is not always necessary for testing and comparing several scenarios. The difference between scenarios could also be represented by using different input data sources for the models. The interaction service merely allows live interaction with the virtual world such that the Digital Twin becomes 'interactive'.

How the interaction service works

For registering such changes, the interaction service exposes a simple API that accepts changes in a certain format. We propose to use the format below. Other standard formats for representing changes (such as JSON patch) are under investigation.

All the changes are registered into a database along with the scenario identifier. The interaction service provides endpoints for retrieving these changes per scenario ID and starting from any given point in time. This allows clients to reconstruct the hypothetical world corresponding to the scenario and run experiments with it. This way, scenarios can be persisted.

There is of course also the possibility to reset a scenario by cleaning up all changes. This reverts the scenario to the baseline state without changes.

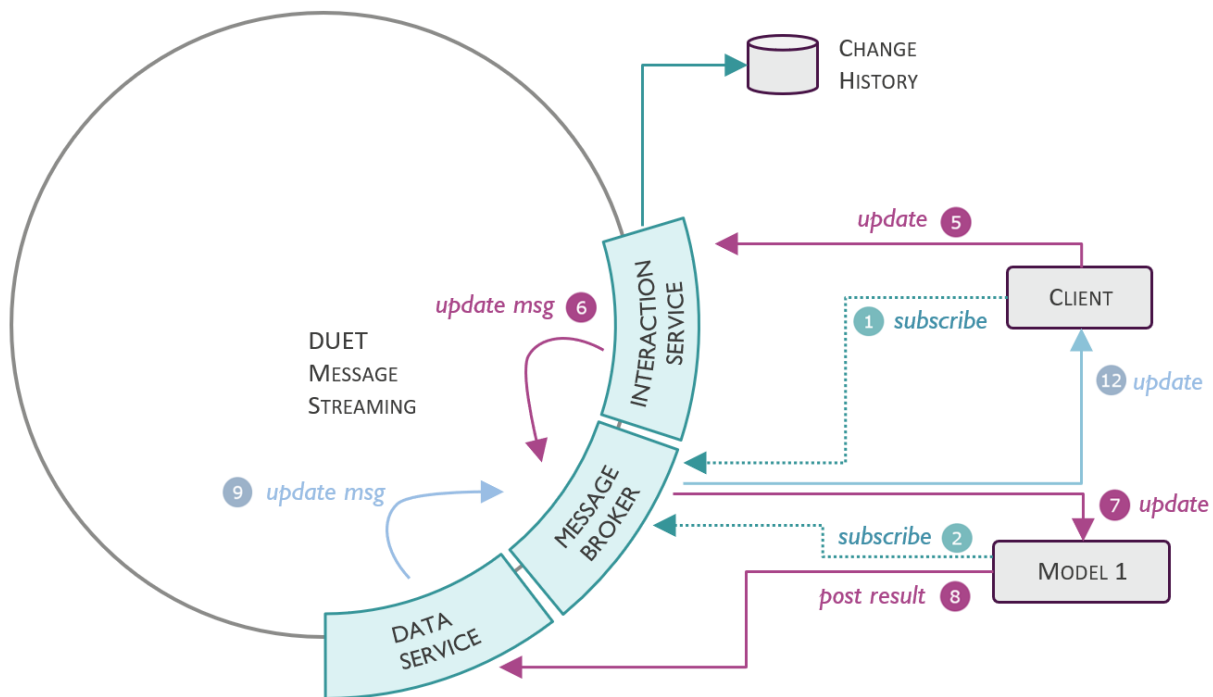
Note:

Before running an experiment, the models must see to it that they are in sync with the changes made in a scenario. This means that before running actual experiments, there needs to be a start-up phase in which a model checks the interaction service for changes that apply to any of its inputs and sync with those.

Aside from keeping the changes in store for later use, the interaction service also connects to the Message Broker to publish the changes, adding metadata such as the scenario ID. The service also allows sending a deliberate batch update message, publishing changes after a point in time as a batch. This gives the other components the possibility to respond appropriately to changes by subscribing to these events with the Message Broker.

Example interaction dynamics

The diagram below explains how interactions work in a simplified way. A more complex version of this diagram can be found in [Addendum 1: Extended interaction flow](#).



- 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.
- 5 **Client 1** sends an update to the **Interaction 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 **Model 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.
- 12 The **Client** is notified of changes to the data and responds appropriately.

2.3 Example Scenario

We generalize some relevant use cases to support the technical discussion. This drives the need for cases and experiments as described above. These cases build functionality to support the [pilot epics](#) : G1, G2, G3, G4 and G5. Summarized, they require policy makers to do what-if analysis, publish the resulting data and allow citizens to give feedback on scenarios.

- A **city administrator** can create a **case** to allow the grouping of several scenarios that each reflect a hypothetical change in the city. (Feature already implemented)
- A **city administrator** can create and manage multiple **scenarios** within a case to compare the impact of different hypothetical outcomes.
- A **city administrator** can create a new scenario from an existing one, copying the entire scenario configuration.
- A **city administrator** can use different models in combination within a scenario to assess cross-domain impact of the changes made in a scenario.
- A **city administrator** can link datasets resulting from what-if analysis to a scenario as result
- A **citizen** can switch between scenarios to be able to assess differences. Doing this, displays the relevant scenario results

Note: see also section [2.4 Scenario Orchestration](#) for a schematic description of these steps.

The specific end-to-end scenario we are looking to support is the following. Given a specific street network configuration in a city, we want to provide insight to city planners and other experts on the effects of making changes in the street network configuration such as closing down one or more streets or changing traffic flow directions in several streets. The effects are measured in terms of traffic intensity and air quality as a result of changes to traffic intensity.

Setting up such an experiment requires several steps:

- Set up the scenario, and add the necessary data sources, in this case
 - a model of the street network of the city,
 - a traffic intensity data source
 - an air quality data sourceThis will make all the data sources available as layers for visualization as well
- Adding a reference to the Traffic model and specifying that it needs to use the street network data source as input, and a traffic intensity data source as output
- Providing the necessary configuration for the Traffic model
- Adding a reference to the Air Quality model and specifying that it needs to use the traffic intensity data source as input and the air quality data source as output
- Providing the necessary configuration for the Air quality model

By configuring everything as such, the scenario can be started, instructing the agents of the different models to start listening for updates related to this scenario. The orchestration is detailed further below in this document.

We are piloting this scenario for the city of pilzen but we hope to apply the same model in other cities too, using different models in the process.

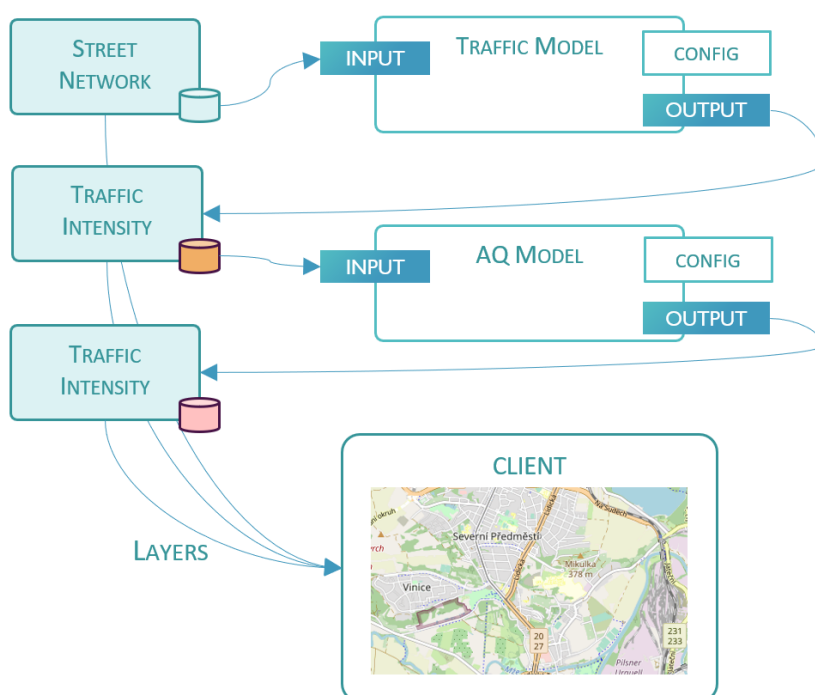


Figure: How a scenario wires up data sources and models

Note:

The configuration property of a model is also deemed part of a scenario. It may indeed be possible to use scenarios to explore and evaluate the outcome of the same model with the same input data but using a different config. Similarly, different models could also be compared by applying the same input data and comparing the outcomes.

2.4 Scenario Orchestration

In this section we discuss how the different components interact end-to-end. For the purpose of this discussion, the entire process has been divided into different independent parts:

- Step 1: setting up the use case and the scenario
- Step 2: starting (and stopping) a scenario
- Step 3: (a) running experiments with batched updated (b) running experiments with streaming updates

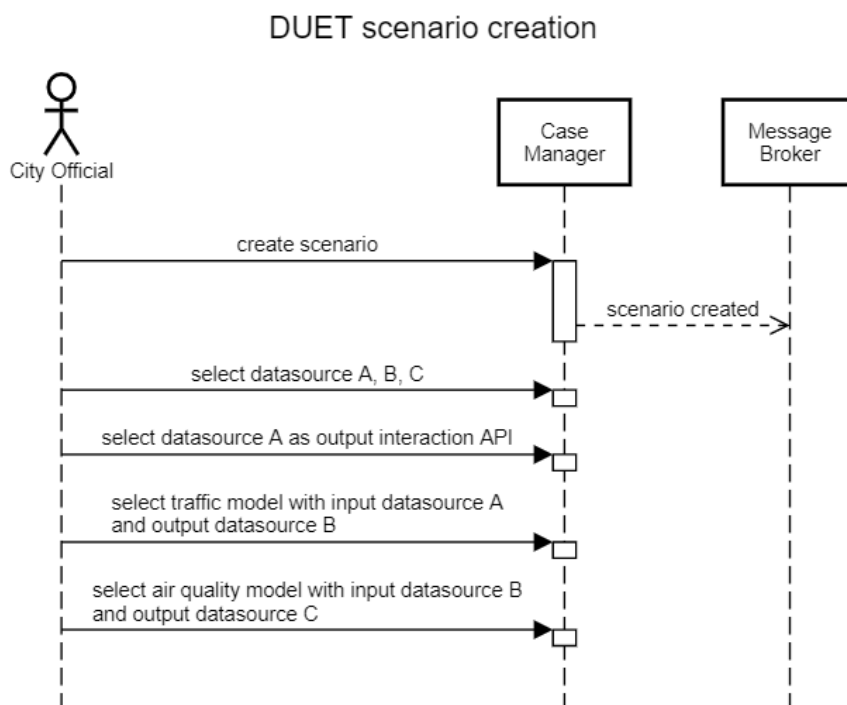
We discuss each of the steps below.

STEP1: Setting up the case and the scenario

An expert user (for instance, a city planner) can create scenarios within a case. With the scenario, he can select data sources and models to run. When calling a model, the required input and output data sources need to be selected. Additionally it is possible to specify a certain configuration for the model.

Implementation note:

The case and scenario management service are work in progress. The open beta release will most probably not allow configuring a scenario as described below and the configuration of what models and data sources to use will remain hard-coded in the model agents for now.



Precondition: case has been set up

The flow as depicted above describes the steps needed:

1. Create a scenario within a case (case assumed to be there)
2. Upon creation other components are notified of the scenario creation
3. Data sources are selected from the data catalog to include in the scenario
4. The models are selected as well and their parameters are configured:
 - a. Input data sources (to be selected from the earlier selected data sources)
 - b. Output data sources (can be newly created data sources)
 - c. Model specific parameters and config

STEP 2: Starting (and stopping) a scenario

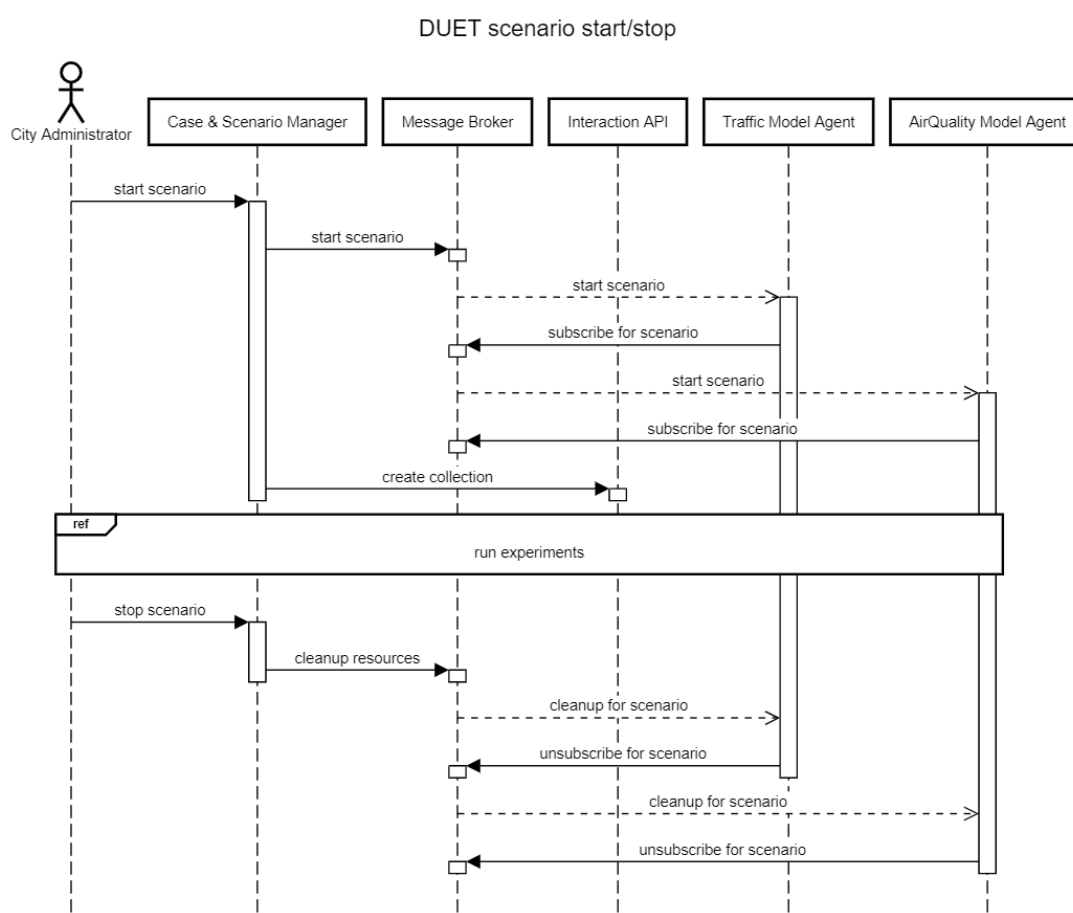


Figure: describing the effect of starting and stopping scenarios.

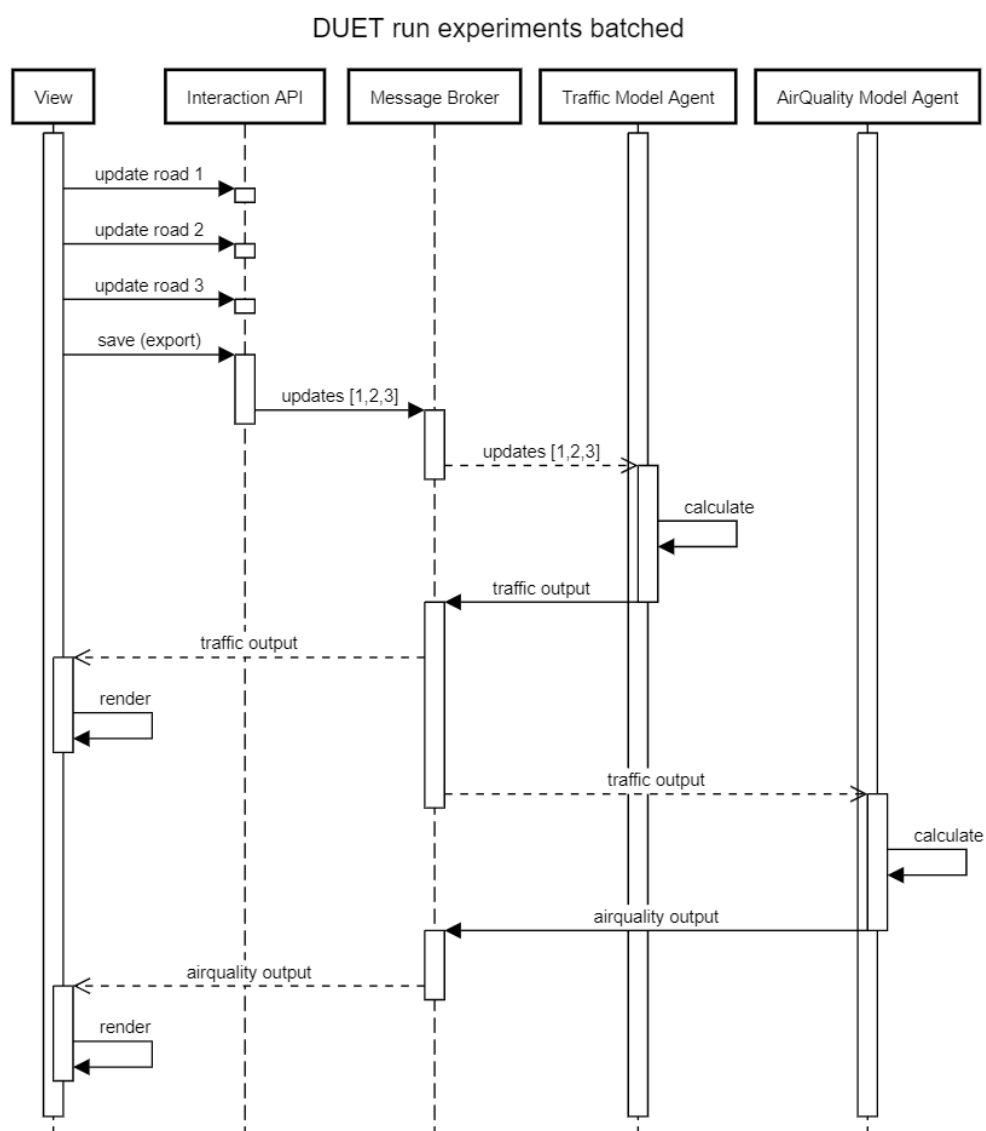
Precondition: case and scenario are set up and fully configured.

The start of a model instructs the model agent to subscribe to messages for a certain scenario. The instruction can be sent upon opening a scenario or as a consequence of a deliberate action of the user.

Starting and stopping the scenario also prevents the model provider from having to keep resources ready and reserved all the time until a client decides to use it. It also provides a handle for the client to cut short a model run after it is started and allows the model agent to release reserved resources.

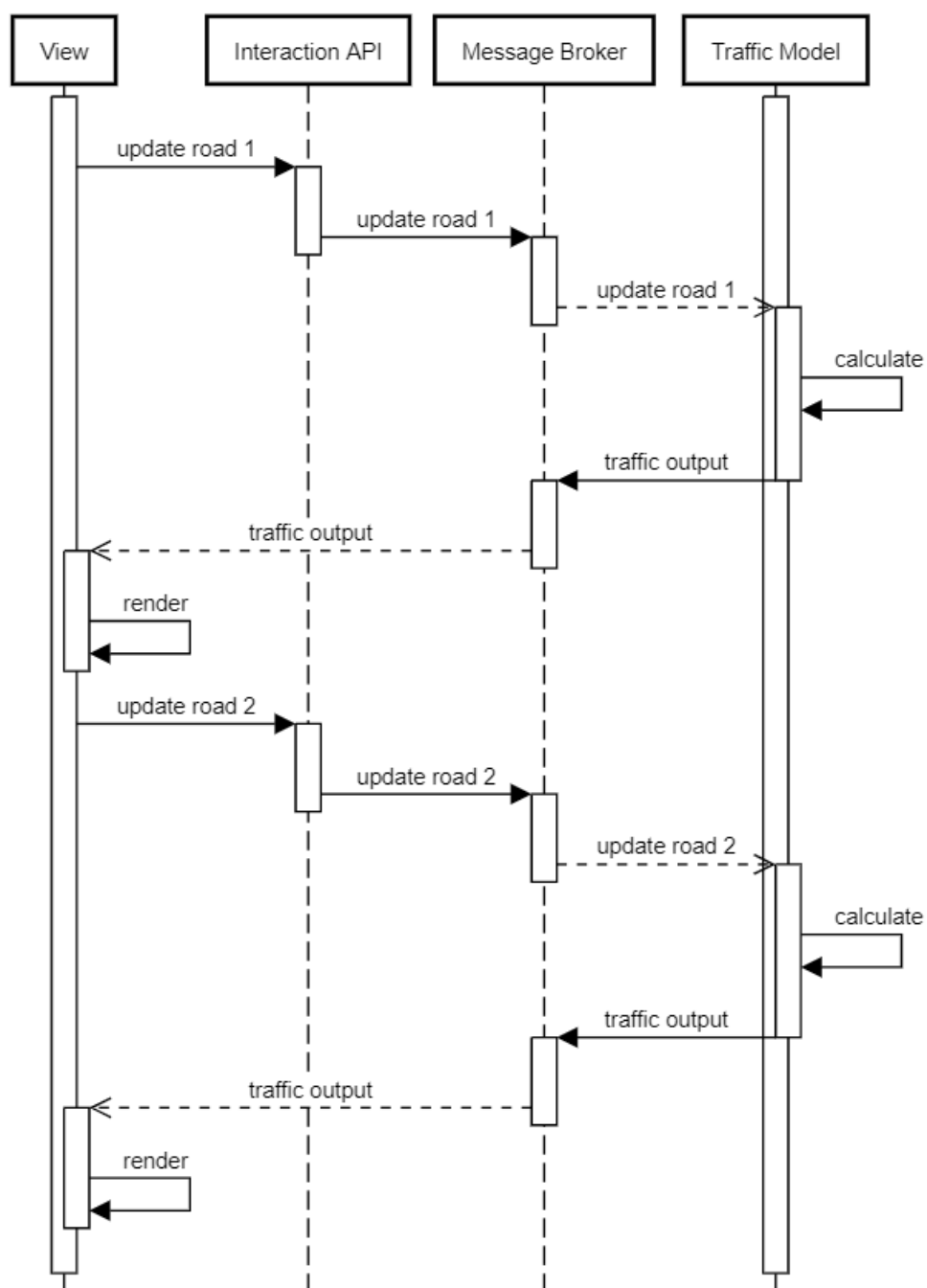
Implementation note:

Starting and stopping models is also not a requirement in the current simplified setup of DUET. As such, the starting and stopping of models as they are integrated now will not be supported in the open beta release. This feature is pushed to a later release.

STEP 3a: Run experiments with batched updates

STEP 3b: Run experiments with streaming updates

DUET run experiments streaming



3. Asset Catalog & Data Brokerage

In the previous sections we explained that the data sources and the models needed to be added to a scenario. This is done by selecting them from the **data catalog** and **model catalog** respectively. Together we call these two components, the **asset catalog**.

3.1 Data Catalog

The data catalog component was discussed in detail in the previous version of this deliverable (D3.8) and most of the ideas and principles set forth by that deliverable remain unchanged. We briefly recapture the features that we envision to be included the data catalog:

- Data source registration and classification
A central register registering data sources, their metadata, the location where they can be found and the technical and semantic standards in which they are available.
- Inventory management
The possibility to add/remove/modify the data sources using an open API in alignment with the access restrictions
- Discoverability
The possibility to find relevant data sources according to a harmonized ontology by using a knowledge graph
- Access management
The possibility for data publishers to control access to their data for 3rd party users
- Brokering
The possibility to access data across the DUET system in a way that is independent of the technical and semantic standards used by the publisher and in alignment with the access restrictions
- Mapping data
The possibility to map between data schemas and formats when accessing the data via the brokering interface. Mapping could be a different type of asset that can be reused across different cases, e.g., when mapping between known standards.

For a more detailed discussion we refer to D3.8.

Indicative screenshots from the data catalog UI for its main functionalities are provided in the following figures.

Add Datasource

Search...

Filters ▾

Type
Select... ▾

Publisher
Select... ▾

Region
Select... ▾

Category
Select... ▾

Format
Select... ▾

License
Select... ▾


Subject
Select... ▾


Sort by Select... ▾

athens 3d lod 1 <i>Athens</i> Athens area / Greece Datasets 2021-05-05	gmns cambridge example <i>Cambridge / USA</i> Cambridge / USA 2021-05-05	cityflows sensors <i>Flanders</i> Antwerp area / Belgium 2021-05-05
cityflows proximus_locatio... <i>Flanders</i> Antwerp area / Belgium Device/Thing 2021-05-05	traffic_3d_gent_at_10 <i>Flanders</i> Traffic for Gent / Belgium at 10:00 Services 2021-05-05	metro_stations <i>Athens</i> Athens area / Greece Services 2021-05-05
pc_plzen_part4 <i>Plzen</i> Plzen area / Czech Republic 2021-05-05	air quality <i>Plzen</i> Plzen 2021-05-05	air pollution 3d <i>Plzen</i> Plzen area / Czech Republic Datasets 2021-05-05


Figure: Datasources list and search parameters


Datasets/ athens 3d lod 1





 Description

Athens area / Greece


 Tags


 Metadata

Region	Athens
Subject	Environment
Publisher	VCS
Format	
License	
Date issued	2021-05-05


 Connections Specs

Type	WEBHOOK
Datasource Url	https://duet.virtualcitymap.de/alpha/datasource-data/f585f924-88a0-4b3b-9602-198b6037d5d1/tileset.json
Client ID	
Schedule	

Figure: Datasource details

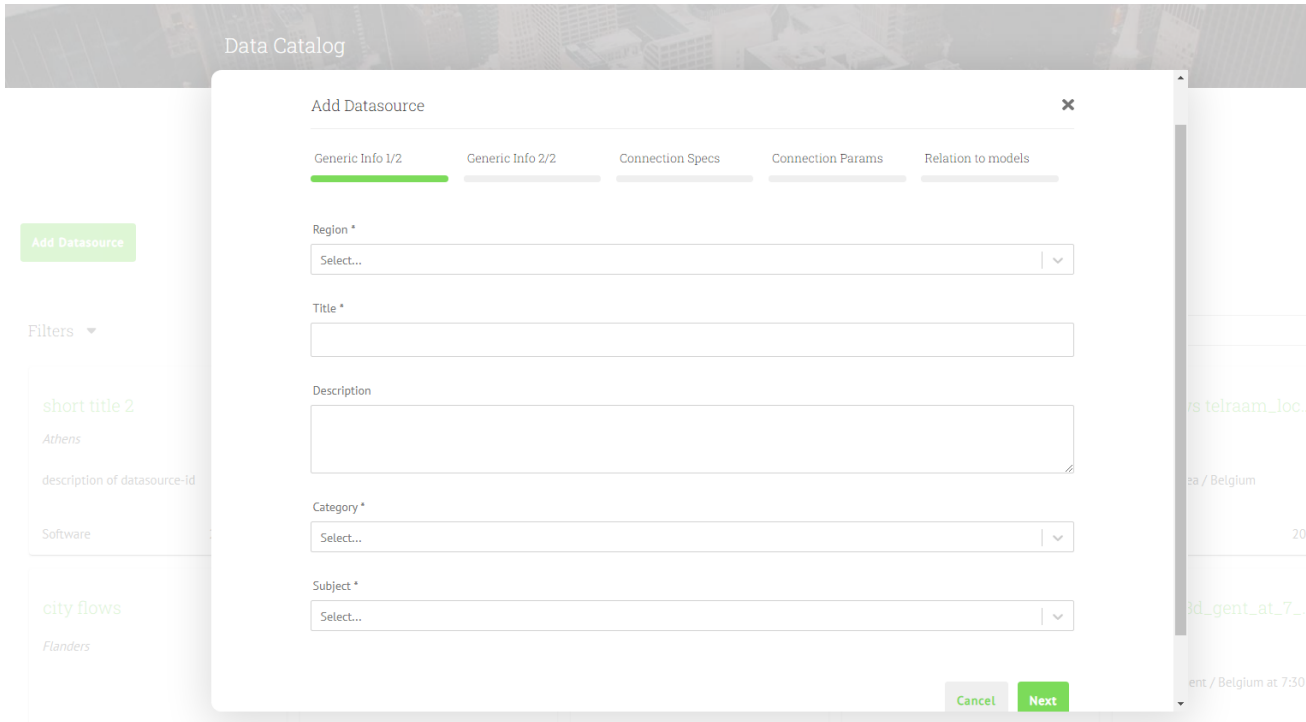


Figure: Add datasource

3.2 Model Catalog & model signature

Model metadata

Just like the data catalog, the model catalog contains an inventory of models that have been made available by DUET users. The model catalog is as such an inventory of available models along with metadata describing all aspects of the model. Most notably, the model signature details how the model can be used in conjunction with data sources and how these data sources are expected to be formatted and what schema they should follow.

The first informal version of an information model to capture such metadata is given below.

```
{
  id: "some-model-identifier",
  name: "model-name",
  description: "full-description",
  more-info: "url",
  model-location: "url",

  input-parameters: [{
    name: "some-parameter-name",
    description: "description",
    format: "serialization-format",
    schema: "data-schema",
    protocol: "protocol",
    cardinality: "? (optional), + (more than one), a positive integer"
```

```

    }],
    output-parameters: [{
      name: "some-parameter-name",
      description: "description",
      format: "serialization-format",
      schema: "data-schema",
      protocol: "protocol"
    }],
  },
  configuration: [{
    key: "setting-name",
    value: setting-object
  }],
  provider: {
    id: "some-duet-organization-identifier",
    name: "organization-name"
  },
  contact: {
    id: "some-duet-contact-identifier",
    name: "full-name",
    email: "email-address"
  }
}

```

Figure: Proposed model metadata information model

The information model needs to be refined further, aligned with existing schemas and ontologies where possible and eventually formalized in a schema and ontology of its own.

Besides the obvious fields id, name, description and so on, the most important parameters that define the model's signature are:

- model-location: the location of the model agent's API on the web making the model reachable through a DUET compliant REST API;
- input parameters: specifying what parameters there are, in what cardinality and what schema and format are expected by the model. We could also specify a protocol (or a list of protocols) that the model can use to access the data;
- output parameters: specifying one or more output data sources where the output needs to go and the format and schema that output data will adhere to. Again here, the supported protocols for publishing the data may be included as an extension;
- configuration: specifying configuration parameters for running the model. This could be any model-specific extension of a basic parameter set that allows defining a time range and a geographical boundary for instance.

Possible extensions are metadata fields about data resolution and thresholds, data quality attributes for input and output parameters and so on.

Model calling

As discussed in section [2.1 Cases, Scenarios and Interactions](#) models can be used by adding them to a scenario and specifying how they should be activated. We refer to this principle as model calling. This means

that the scenario will allow the user to specify how to connect data sources to models. This can be done with the following proposed information model.

```
{
  model-id: "some-model-identifier",
  credentials: "model-credentials",

  input-parameters: [{
    name: "some-parameter-name1",
    data-source-id: "some-data-source-id1",
    protocol: "protocol1",
    credentials: "datasource-credentials1"
  }, {
    name: "some-parameter-name2",
    data-source-id: "some-data-source-id2",
    protocol: "protocol2",
    credentials: "datasource-credentials2"
  }],
  output-parameters: [{
    name: "some-parameter-name3",
    data-source-id: "some-data-source-id3",
    protocol: "protocol3",
    credentials: "datasource-credentials3"
  }],
  configuration: [
    time-range: { begin: "timestamp", end: "timestamp" },
    geo-boundary: { ... }
    ...
  ]
}
```

This way, all the information that is needed by the model to perform its run is available in the scenario, including the scenario identifier that provides a safe scope for model execution. The scenario identifier can be used to prevent messages and data from getting mixed up between different scenarios and cases.

3.3 Brokering Ubiquitous Models

As a last step in the process to fully decouple models from a specific application is then to run them as a service behind the DUET compliant REST API (Model Agent API). This API allows you to start a model by sending it a scenario id and the model calling snippet above.

The scenario ID provides an isolation scope for model runs that prevent data and messages from getting mixed up between different model runs from different cases and scenarios but it also tells the model what messages it needs to listen to coming from other components and data sources that may drive the orchestration of the model. This implies that in our approach, some details of the orchestration of the model run is left to the model provider and is achieved by subscribing to incoming messages and publishing other messages in return.

That also means that combining models in a scenario can only be achieved by binding the output of one model to the input of another (see section [2.3 Example Scenario](#)). This approach resembles the one taken by Cloudflow Streamlets (see picture below), where orchestration is achieved by connecting input and outputs of different components.

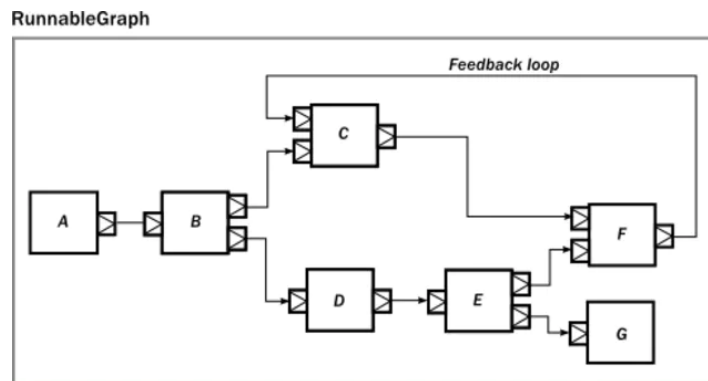


Figure: The cloudflow streamlet³ approach resembles the way DUET achieves orchestration.

Once models are being served behind the abstraction of a model agent API, they can be used in conjunction with alternative compatible data sources in other cases and scenarios, perhaps even in other cities and combined with other models.

³ [Developing Streamlets :: Cloudflow Guide](#)

4. Tools

4.1 Data sources, models and the knowledge graph

The main purpose for using a knowledge graph in DUET is for data discoverability. Eventually it may help in mapping as well but it is not a necessity. In order to build our knowledge graph, it is essential that data sources can be mapped onto the general ontology. As discussed in D3.8, this requires a unified ontology, preferably based on existing standards.

At present, we envision this mapping to be done manually using a tool such as TopBraid (see below). This entails that individual attributes of data sources can be labeled with their semantics based on definitions in the Urban Digital Twin ontology. This will allow us to find data sources using general search terms, even if their native formats do not use such terms. Although such a mapping may help to transform these data sources to a common ontology, information model and data format, this remains far from trivial and will still require manual work (see deliverable D3.8, section 3).

Finally, as we will discuss, this technique may also be applied to model discoverability. We will describe models with their signature, which includes a schema for their data sources. Again, this schema can be mapped with tools such as ToBraid (see below) onto a shared ontology enabling us to not only discover data sources but also models that can be applied onto them.

4.2 Supporting model management

No tools exist today that allow us to manage models (simulation models, machine learning models or hybrid models) as a library of assets and link these with their signatures that include schemas. As such we are building a simple version of a model catalog as described in [3.2 Model Catalog & model signature](#). As stated in the introduction, this is still work in progress.

In order to achieve discoverability of data sources and models on the one hand and in order to ensure compatibility between models and data sources on the other, it is important to dispose of the schema of data sources and the signature of models. This will be implemented in a basic fashion in the first version of the system.

A lot of the results of the DUET project come in the form of new insights. One of the most important ones is the need for a schema entity in the asset management service that allows anyone to specify the data schema they used for their data source or that they are assuming when their model uses data. When the schema of a data source matches that of a model parameter, then the data source can be used - provided other aspects of the data match with what the model expects. If not, a mapping may be needed beforehand.

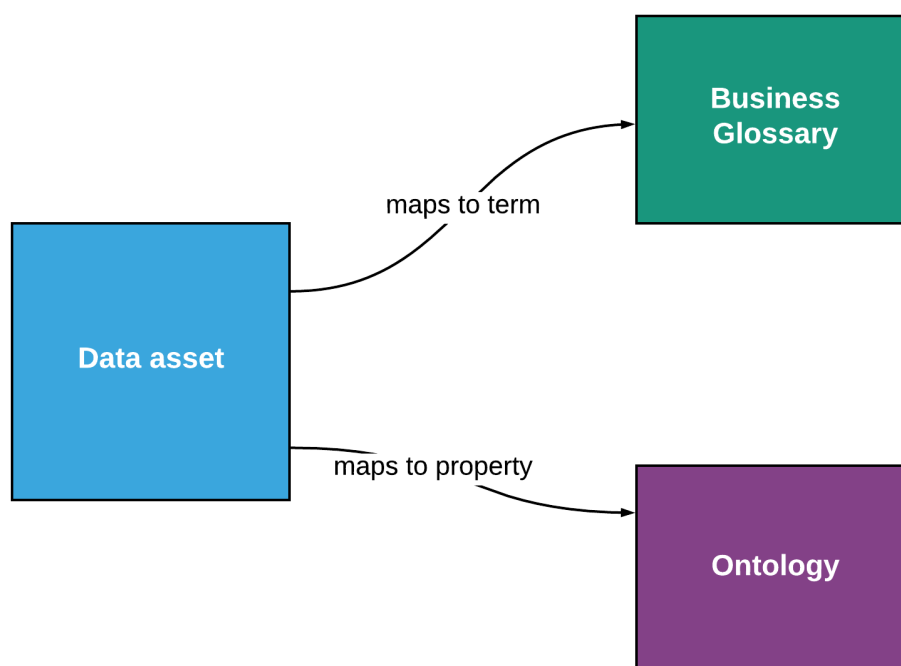
To develop this further, there is the concept of a (refined) schema (see also Deliverable 3.2, section 4):
A refined data schema provides a full and unambiguous specification for the data elements in a dataset by combining a data model, domain specific data constraints and semantics in order to facilitate full technical and semantic interoperability.

These insights provide fertile ground to further develop DUET and the DUET concepts in future projects and initiatives and should be considered valuable output. We also have started an initiative to define a Model

Interoperability standard together with Fiware, OGC, OASC and other stakeholders building on the results and insights of this project.

4.3 TopBraid

TopBraid is an asset governance tool that can be used to create **refined schemas** for **datasets under management**. Refined schema means having a technical schema with type and value constraints for the data of that schema. Three components in TopBraid are used to achieve this goal: data assets, business glossaries and ontologies. The elements in these three components are linked together using RDF links. The sections below explain each of these components and their interactions in more detail.



4.3.1 Data Assets

Data assets model the data sources that we want to bring under management. They capture the technical schemas of those data sources, based on the original schema. For example, when bringing a data source under management with a json schema, each element in that json schema will have a corresponding element in the data asset. Each element in a data asset is then linked to business glossaries and ontologies through their **maps to term** and **maps to property** properties respectively. Mind that creating these data assets is fairly tedious work, since TopBraid does not provide an option to import a json or xml schema and convert it to a data asset. All of that has to be done manually.

4.3.2 Business Glossary

A business glossary contains the terms and concepts used in the data assets defined in non-technical language. Through the business glossary, non-technical stakeholders can discover the data sources. This is done by querying the **maps to term** link between data asset and glossary using SPARQL.

4.3.3 Ontology

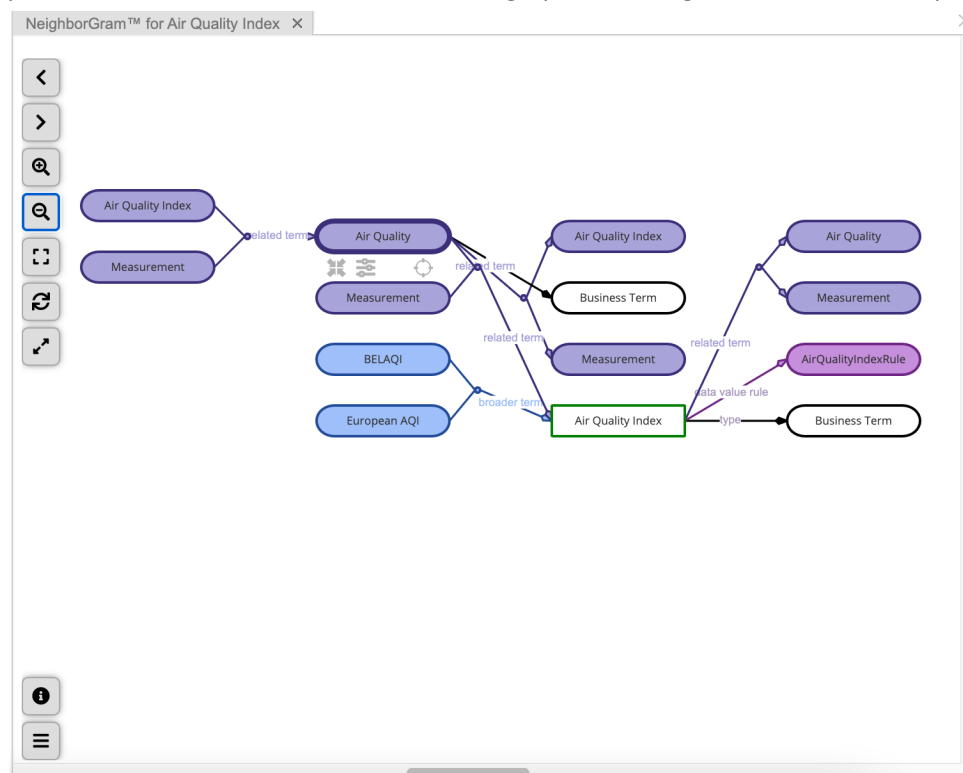
Ontologies consist of SHACL classes that define the concepts used in data assets. Through the **maps to property** link between data asset and ontology, we can apply SHACL constraints to data sources (e.g: a data

element that represents a percentage should satisfy $0 \leq \text{data value} \leq 100$). These constraints can be enforced at ingestion time of new data.

Existing ontologies can be imported into TopBraid, but all OWL properties are converted to SHACL during import. However, there is no one-to-one mapping OWL to SHACL, so some OWL properties will be lost.

An advantage that results from working with the **maps to ...** links is that abstractions can be made over disparate data sources. Instead of reasoning about air quality data from a single provider, we can reason about air quality data in general, since querying the links between data asset and glossary/ontology will return all the relevant data sources.

Topbraid also provides flexible visualizations of the RDF graphs resulting from the three components:



Lastly, the RDF data of these three components can be exported in various formats from TopBraid, so that it can be used in the actual asset registry.

By exporting the RDF data from TopBraid, it may become possible to enable discovery of data sources and models based on a shared ontology.

4.4 APIAPI

Another project worth mentioning is the Belgian APIAPI project⁴. It provides an API abstraction layer that maps the data sources and APIs of smart city data publishers onto a single data model, such as the Flemish OSLO standards.

⁴ [APIAPI | Foresight Gent](#)

This open source software makes data management for city applications easier. The need became clear when trying to port applications and components from one city to another. Software may have a reusable structure (so that you can immediately start using it in another city), the precise data sources you want to use in the software are always specific to one city. Even if the application is generic, one still has to connect the data sources, usually in a manual way. Data can be of various origins (a self-built data stream based on Internet-of-Things sensors, a data stream provided by a start-up, archived data, etc.). The APIAPI helps to set up uniform data streams from these various sources. In this way, you can combine disparate data and provide the right structure to an application. APIAPI allows to accelerate the introduction of reusable solutions in cities, and thus also in urban digital twins.

Although it does not offer the advanced data management features of TopBraid and does not allow exporting a knowledge graph, it is a low-touch pragmatic solution to bring uniformity into the chaos of smart city data.

5. Conclusion

Over the past months, the technical team has been focusing on the delivery of the Open Beta. Where the Closed Beta was more a proof-of-concept for the overall architecture, the Open Beta release is striving to be closer to the decoupled digital twin of components that was intended by the T-Cell architecture. Perpetual insight has caused the architecture and approach to evolve a bit in some areas as discussed in this document.

Current focus

This delivery focuses on the interplay between the models, the data sources and the client. We are proposing a conceptually sound and sufficiently powerful design that will allow us to combine models in cross domain cases at will without the need for a technical intervention. we cannot stress enough that this implies a few things though:

1. The management of data sources as assets with enough metadata that allows a good understanding of the usefulness of the data sources by their consumers (i.e., data governance);
2. The management of models as reusable assets with sufficient metadata for users of the model to understand its intended use and its limitations;
3. Common agreements on data schemas and semantics to facilitate compatibility between data producers and data consumers such as models, assisted by reusable mappings that can also be managed as assets and that lower the barrier for data reuse;

We point out that these assumptions are the foundations of data spaces. A data space is a federated ecosystem of systems that aims to share and exchange data, algorithms, applications and components that is framed by shared agreements, standards and technologies. IDSA provides a reference architecture for such dataspace. It mentions terms such as App Store (which can be seen as a catalog of algorithms, client software, etc.), brokers, connectors and so on.

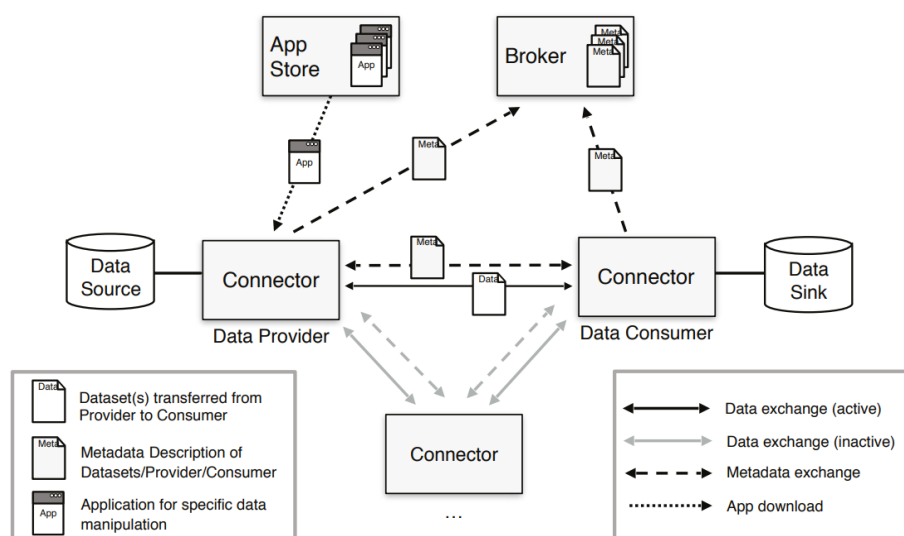


Figure: The interaction of technical components in the IDSA architecture

Further work

Having gathered many new insights and made headway into implementing a digital twin marketplace, we consider the DUET project (and this deliverable) not to be an end point. It is merely a step in the direction of European Urban Dataspace: dataspace according to the principles set forth in IDSA and Gaia-X that focus on urban data. Although DUET delivers on its promise to deliver a first version of a framework where models and data from different providers can work together in a seamless and extendable way, quite some ground remains uncovered. The creation of further abstractions and standards to make models and other apps truly interchangeable and the realization of common vocabularies for smart city domains are of particular interest. Steps in that direction have been taken and we remain eager to progress further in this domain.

Implementation: current status and commitments

The upcoming open beta release of DUET will include many of the newly introduced components. This release is focusing on supporting multiple smart city cases across more geographical areas, also integrating new models.

However, as discussed in the introduction, the integration of models is not yet happening entirely by the principles set forth in this document. The next step in our project is to make that integration happen. This means:

1. Introducing the model catalog as part of the asset catalog (this is work in progress)
2. Allowing the configuration of cases and scenarios from the client in the case manager
3. Configuring model calls statically at the level of a scenario (this is work in progress)

This will be a good outcome to start looking at a standard technical and semantic interface for providing ubiquitous model infrastructure, making models more open, reusable and portable across systems and realizing true urban digital twin marketplaces.

6. References

CKAN

The open source data portal software

<https://ckan.org/>

OSLO

Open standards for linking organizations (dutch)

<https://overheid.vlaanderen.be/producten-diensten/oslo>

VLOCA

Flemish open city architecture

<https://vloca-kennishub.vlaanderen.be/>

INTERNATIONAL DATA SPACES ASSOCIATION

Reference architecture model

[IDS Reference Architecture Model 3.0 \(internationaldataspaces.org\)](https://internationaldataspaces.org/IDS-Reference-Architecture-Model-3.0/)

TOPQUADRANT

TopBraid EDG

[TopBraid Enterprise Data Governance — TopQuadrant, Inc](https://topquadrant.com/topbraid-enterprise-data-governance/)

LIGHTBEND

Cloudflow Streamlets

[Cloudflow streamlets :: Cloudflow Enterprise Features Guide \(lightbend.com\)](https://lightbend.com/cloudflow-streamlets/)

IDSA

[International Data Spaces | The future of the data economy is here](https://internationaldataspaces.org/)

GAIA-X

[GAIA-X - Home \(data-infrastructure.eu\)](https://gaia-x.eu/)

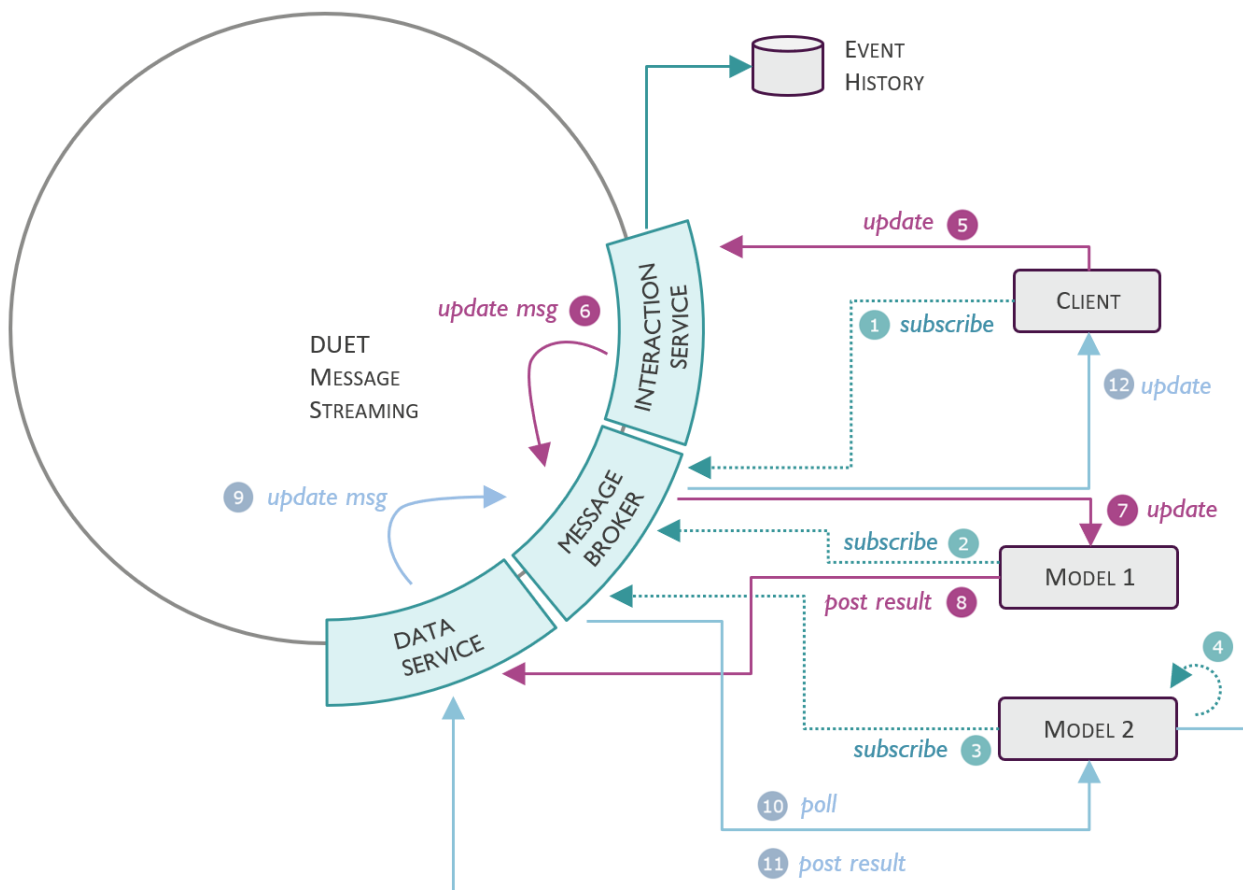
APIAPI

A visual API mapping tool

Foresight Ghent

[APIAPI | Foresight Gent](https://apiapi.foresight-ghent.com/)

Addendum 1: Extended interaction flow



- 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.
- 5 **Client 1** sends an update to the **Interaction 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 **Model 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 and responds appropriately.