Deliverable

D4.3 DUET Simulation models

Project Acronym:	DUET	
Project title:	Digital Urban European Twins	
Grant Agreement No.	870697	
Website:	www.digitalurbantwins.eu	
Version:	1.0	
Date:	09.06.2021	
Responsible Partner:	TNO	
Contributing Partners:	IMEC, KUL, VCS, P4ALL, UWB	
Reviewers:	**Internal** Willem Himpe (KUL) Karel Jedlicka (UWB) **External** Pieter Morlion Andrew Stott	
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	17.03.2021	Max Schreuder	TNO	Initial structure
0.2	28.04.2021	Philippe Michiels	imec	Clean up structure and align with proposed outline + add model schemas
0.3	20.05.2021	Philippe Michiels	imec	Completed section 3
0.4	25.05.2021	Max Schreuder	TNO	version for external review
0.5	26.05.2021	Walter Lohman	TNO	added draft conclusions
0.6	26.05.2021	Pieter Morlion	MORE LION	external review
0.7	01.06.2021	Andrew Stott	AIV	external review
0.8	02.06.2021	Max Schreuder	TNO	executive summary added, updated introduction, processed comments
0.9	04.06.2021	Max Schreuder	TNO	ready for review
1.0	09.06.2021	Max Schreuder	TNO	final version

Table of Contents

List of figures

List of tables

1. Executive Summary

This deliverable covers the implementation of relevant models regarding traffic, air quality and noise. With this demonstrator deliverable and the closed beta version as a PoC, it is shown that an architecture built around the Message Broker in the DUET core is capable of creating an effective chain of models.

The progression on model implementations since D3.3 is described, as well as the message schemes that have been defined to enable the model-to-model connections. The closed beta version of the DUET system currently requires the models to be running to execute the User Stories. The separation of scenarios is done by temporarily introducing a 'scenario id' in the message exchange.

A scalable solution is demonstrated for executing several User Stories. When a user obstructs a road element as a 'what-if' scenario, the system propagates this change to a traffic model. The traffic model calculates the impact on the road infrastructure and signals the DUET system of the data changes. These changes trigger the Noise and Air Quality models to calculate the impact of these changes, respectively. At completion, models signal the DUET core of their data changes. The overall data changes are picked up by the User Interface for evaluation by the user.

Future work includes:

- Adaptation of the Model Agent API (D3.5), which will be responsible for deploying models, data and scenario context on demand, such that users can execute their stories independently of each other.
- Formalisation of how and where (static) data is to be stored and how Data Context Graphs are to be used to register the data changes (dynamic data).
- Current models will need to be expanded to support the data storage mechanisms and the Model Agent API for inclusion in the final DUET system.

2. Introduction

Deliverable D4.5 describes the implementation (method) of simulation models in the demonstrator; the DUET closed beta version. This deliverable covers the ongoing implementation (Task 4.1 Model Visualization (M7-M30)) of relevant models regarding traffic, air quality and noise. The traffic model will produce time-dependent measures such as traffic flows for every road within the considered region for different scenarios. The air quality model and noise model will help visualize the effect of changes in the traffic model. The scope of this deliverable covers the implementation and integration of the simulation models in the DUET closed beta version.

Given that this is a demonstrator report, the technical details of the platform itself are only briefly touched upon. These topics are discussed in more depth in the following deliverables:

- D3.5 describes the general interaction mechanism to connect models to DUET. This principle is also described in more detail in D3.2.
- D4.5 describes the implementation of 3D visualization tools in DUET, which is highly related to model visualization (Task 4.1).
- D4.2 describes data types and data integration in the DUET Alpha Version and outlines data integration as expected to be implemented in the final DUET system.
- D5.1 explains the general system architecture and integration plan,
- D5.2 explains the implementation details of the DUET prototype.

This demonstrator report mainly discusses the results from implementing the following epics:

User Role	Epic No	Epic description
Public servant	G1	**As a** public servant of a relevant department (mobility, spatial planning and environmental department,...) **I want to** see the difference in density of traffic in the area of interest of a scenario where I closed traffic in a set of roads versus the base density, **so I can** assess the impact of changes to the local situation on the traffic in my area of interest
	G2	**As a** public servant of the mobility or environmental protection department, **I want to know** the level and impact on air pollution when certain roads would be closed **so I can** discover causes of air pollution and the impact on citizens well-being in the city
	G3	**As a** public servant of the mobility or environmental protection department, **I want to know** the level and impact of noise pollution when certain roads would be closed, **so I can** discover causes of noise pollution and the impact on citizens well-being in the city

Table 1: epics and user roles related to this deliverable.

These epics provide support for the basic use case of being able to select and inspect a road segment, alter its properties, and have the models respond to the new reality by recalculating (a) the new traffic intensity on all the roads and (b) the predicted noise and air pollution levels. In the closed beta, the integration of models and the visualization of their results is still relatively tightly coupled, but it serves as a proof of concept for the way models can work together, nevertheless.

Chapter 3, High-level data flow, describes the mechanisms in place that facilitate the models running continuously, e.g., how their results are stored and can be processed by each other interactively. It also describes the minimal implementation that is used for the Closed Beta. Chapter 4, Implementation of Models, discusses the progress on model implementations since D3.3 and specifies the message schemes that have been defined to enable the model-to-model connections, using modern standards as they exist at this point.

3. High Level data flow

3.1. Basic flow for what-if scenarios

In simple what-if scenarios, user induced changes in the virtual world (e.g., closing down a street) are reflected in a context database via a system called the Interaction Service. This Interaction Service exposes an API that allows visualization clients to become interactive and notify other digital twin components of any changes made in the client. This way, after the road attributes change, a model run can be triggered - either manually or automatically - that takes into account the new state and can show the impact of the change. In the case of the closed road, it may, for instance, divert traffic and cause congestion elsewhere.

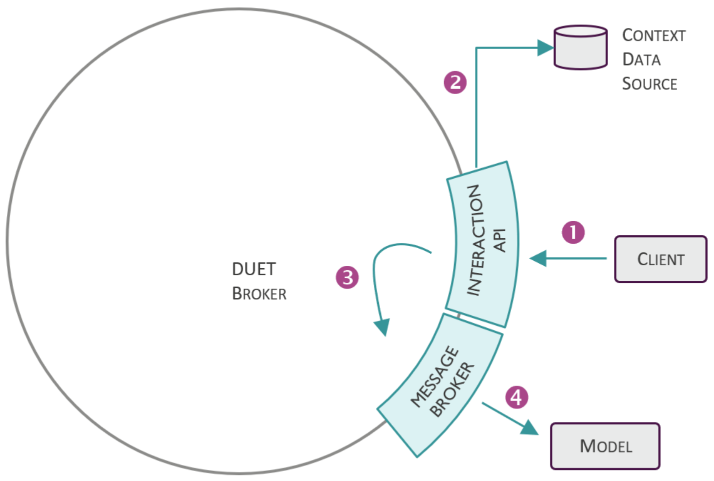

Figure 1: High level message flow between components related to what-if scenarios

Legend figure 1

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.

3.2. Model orchestration

In more complex set-ups, more than one model may be involved, and the models can influence each other. A well-known example is the entanglement of a traffic model and an air quality model as presented by imec and TNO in a digital twin of Antwerp in 2018. This demonstrator technology showed that when traffic was diverted by changing the street layout, a local improvement of the predicted air quality followed as a consequence of feeding the traffic prediction from the traffic model as an input to the air quality model.

DUET aims to take this concept a step further by generalizing the concept and allow multiple models fed by several data sources to interact in a complex set-up. To this end, DUET specifies the Model Serving interface that allows model providers to connect to the DUET core generically. The models also interact with DUET generically through the DUET Data API. As such, they can consume and produce each other's data. Finally, a notification system allows the models to signal changes or subscribe to them. When a change occurs due to user manipulation or as a result coming from a model, other models can respond to that change by recalculation. On the other hand, clients can subscribe to these changes as well and display the new state. The following schematic explains how two models are tied together.

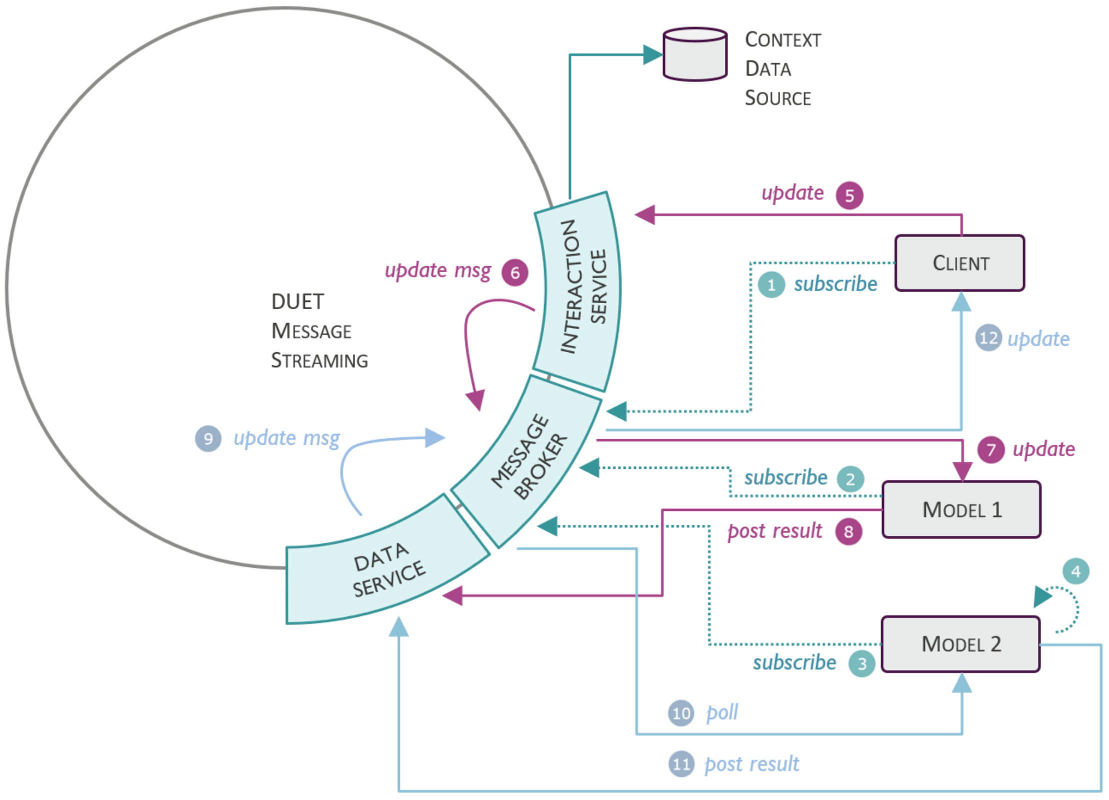

Figure 2: Model connection.

Legend figure 2

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 de 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 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 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).

3.3 Current and future implementation

In the Closed Beta release of DUET, a traffic model, an air quality model and a noise model are connected to the DUET core through the Message Broker Service, which is a notification service that allows the models to send messages to other components.

Currently, the messaging is done by directly publishing to named channels in the underlying message streaming system (Kafka). This needs to be integrated with the DUET data catalog and the yet-to-be-built model catalog to abstract away the Kafka messaging system. This will require a change in the message broker system. Instead of using Kafka topic names, the names of data sources and models will be used instead.

Another component that needs to be added is the Model Service which allows clients to control models. When the Model Service is ready, model agent wrappers can be created for the different models to allow interactions between models and other DUET components.

4. Implementation of models

4.1. Introduction

This chapter describes the progress on model implementations since D3.3 and specifies the message schemes that have been defined to enable the model to model connections.

4.2. Models of DUET overview

4.2.1. Static traffic Assignment

Static traffic Assignment for DUET provided by Plan4All is described in D3.3 (see section 3.1.1. Static Traffic Assignment).

4.2.2. Noise modelling

Noise modelling for DUET provided by Plan4All is described in D3.3 (see section 3.3.1. Noise model).

4.2.3. Dynamic Traffic Assignment Model

Our dynamic traffic assignment model has been implemented in python; there are still some refinement steps to be taken at the time of this writing. It will be available as open-source code within the next few months at https://gitlab.kuleuven.be/ITSCreaLab/mobilitytools . In terms of technologies used, it's an extension to what has been presented for the Alpha version, see https://gitlab.kuleuven.be/ITSCreaLab/mobilitytools/python-traffictoolbox and D3.3, section 3.1.2. Some initial gifs of the output are displayed in Figures 3-6 on the following pages.

There are still some errors related to the equilibration of route choice, which will be mitigated after the closed beta version. What is available for the closed beta version is a simple assignment to shortest paths. This means that travellers do not reroute even though they may experience a substantial delay on their current route. The benefit of this simplification is that traffic states can be calculated quickly, which lends itself to fast integration tests, and it is quicker to validate that the communication between the different components is working correctly.

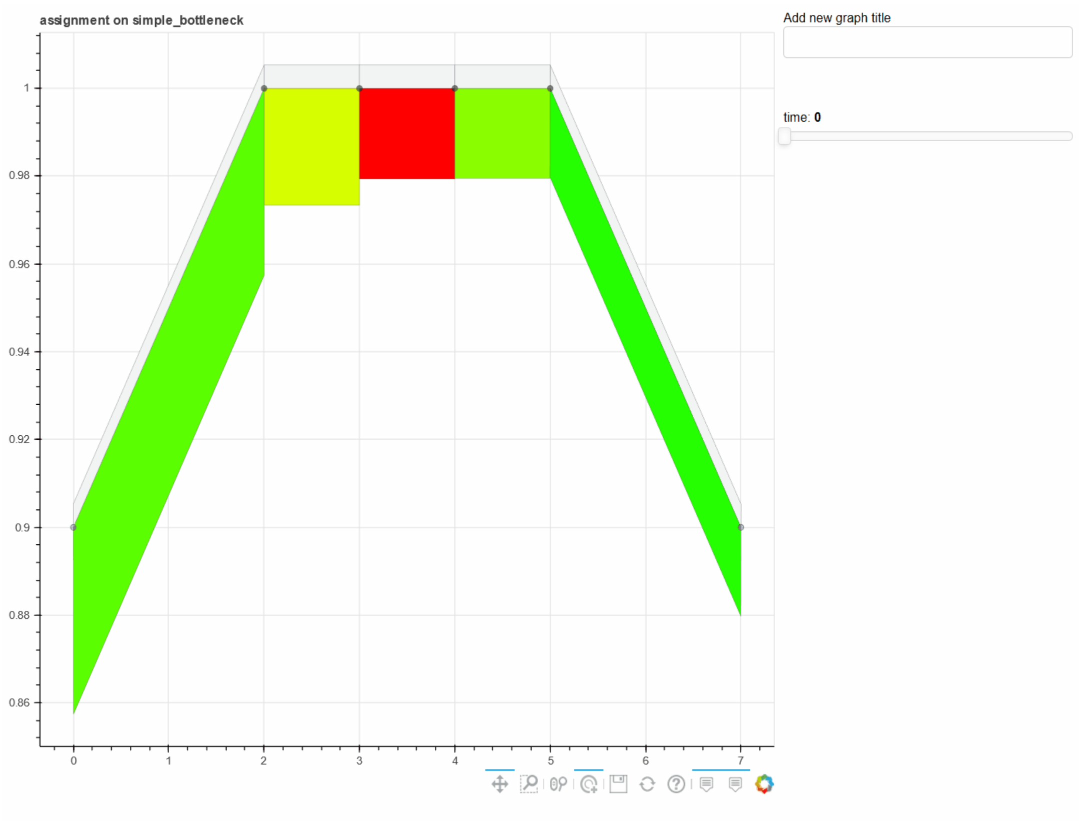

Figure 3: Simulation of a simple bottleneck.

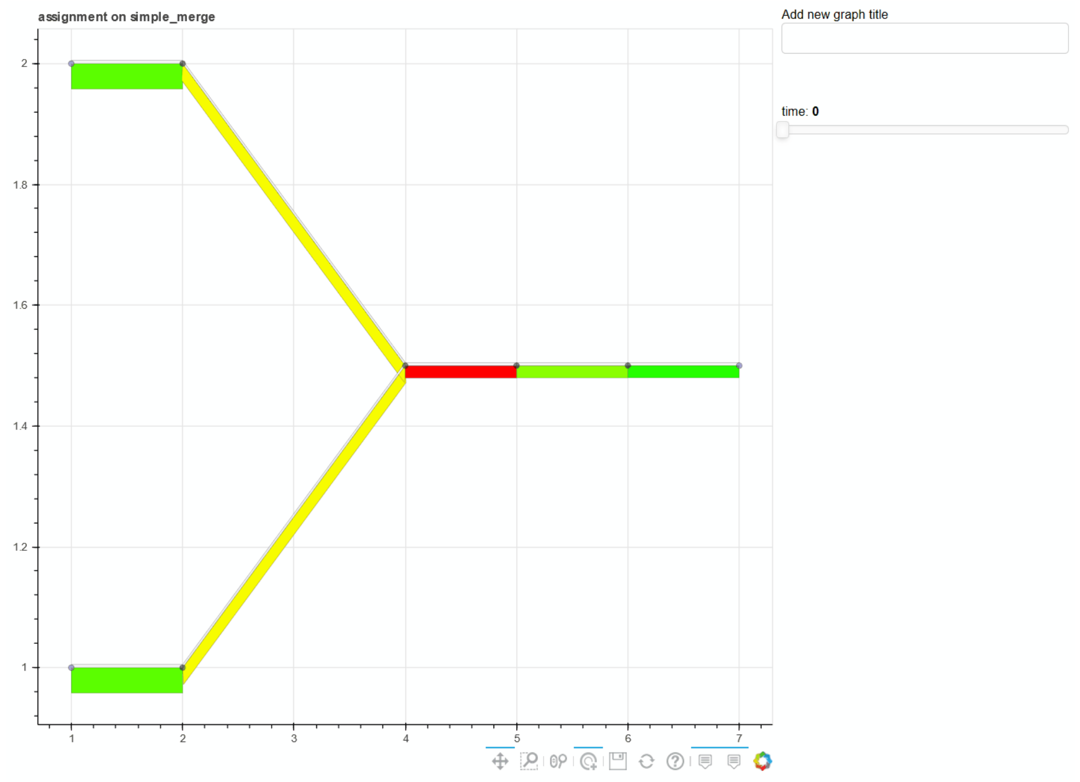

Figure 4: Simulation of a simple merge.

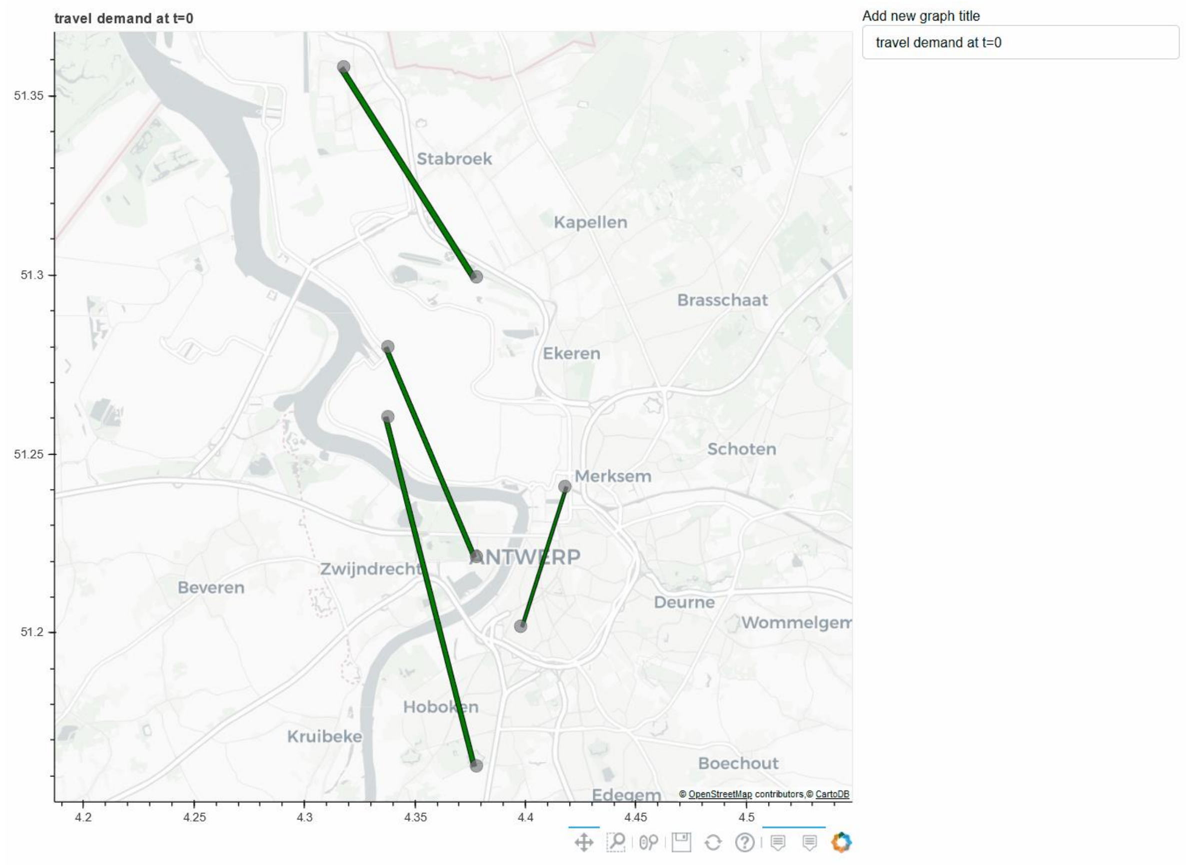

Figure 5: A randomly generated mobility pattern associated with the traffic state just below.

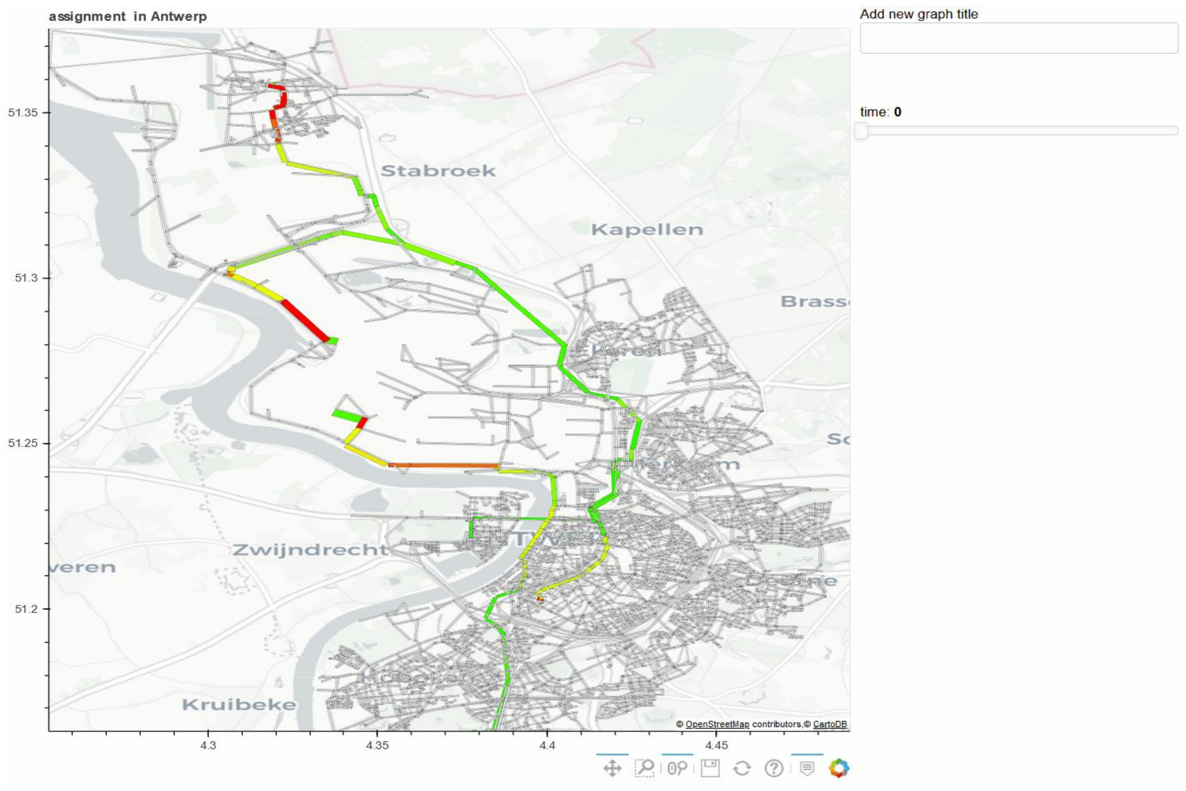

Figure 6: Results of a dynamic assignment in Antwerp visualized on a map.

4.2.4. Air Quality Emission model

The Air quality emission model for DUET provided by TNO is described in D3.3 (see section 3.2 Air quality emission model). Concentrations are calculated for points placed such that a map of Air pollution concentration can be generated. Input for the model are static emission data, derived from a pre-processing stage using the Lotos Euros model (LOTOS-EUROS Air quality modelling and emissions | TNO) and dynamic emission data derived from (dynamic) Traffic data. Calculation points are used to generate the Air pollution map representing yearly averaged concentrations. The Air Quality model is written in C++ and uses Nvidia GPUs with CUDA to improve performance.

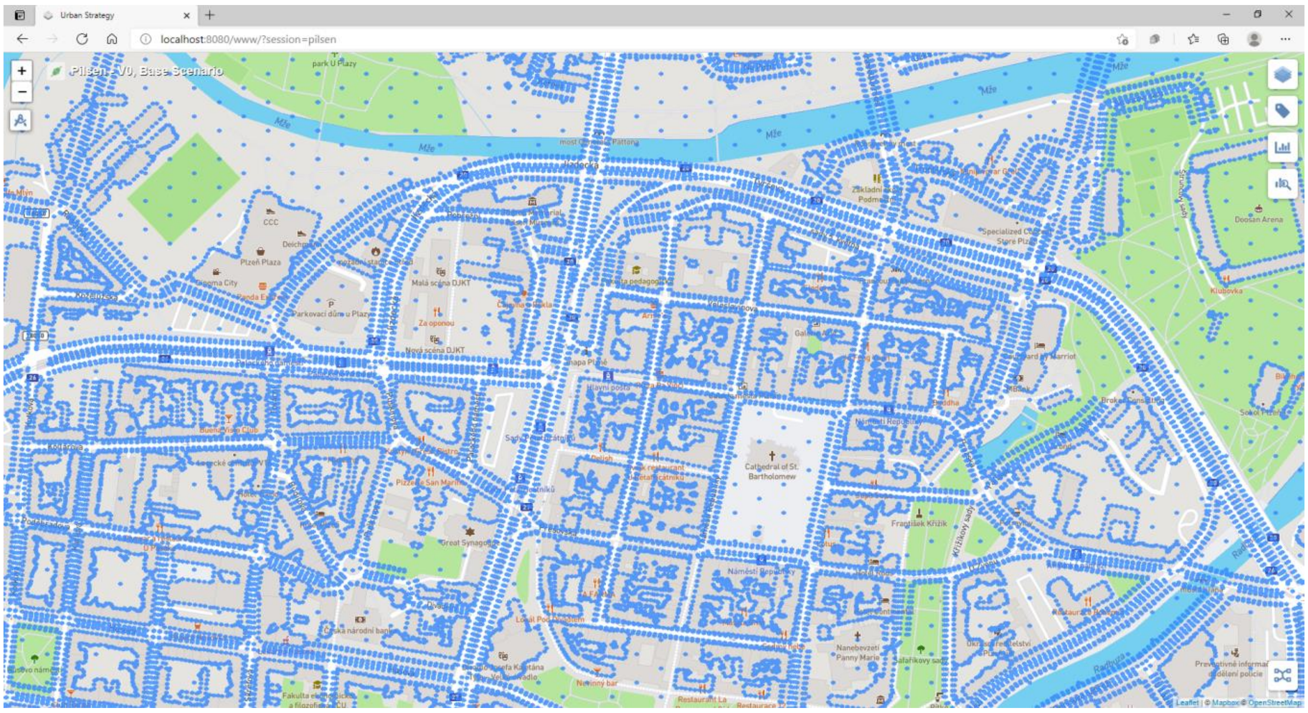
Urban Strategy
localhost:8080/www/?session=pilsen
Pilsen - V0, Base Scenario
Cathedral of St. Bartholomew

Figure 7: Air model calculations points.

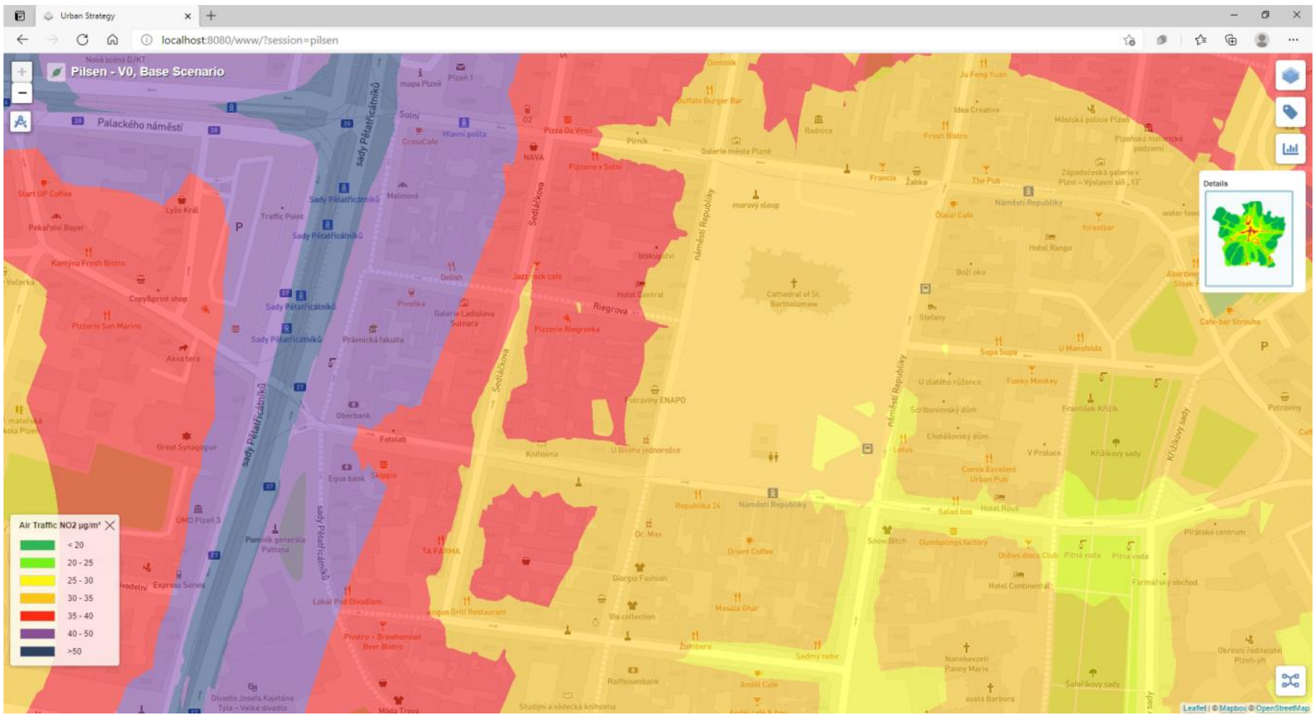

Figure 8: Air model concentrations.

4.3. Model Agent APIs

The following example represents the schema structure for context update messages. It is described in more detail below. The message structure was chosen to be easily parsable.

```
{
   "topic":"",
   "clientid":"",
   "payload": ""
}
```

Message Header:

- topic: identifier of the topic
- clientId: source of the update
- payload: link for the message content - definition depends on the type of data sent

Example for a message from interaction-client to the traffic models responsible for the road network in Pilsen:

```
{
   "topic":"road-network-pilsen",
   "clientid":"interaction-client",
   "payload":                                                                        {
        "url":"http://duet.com/payload.json",
        "scenarioid":"scenarioidhash"
        }
}
```

4.3.1. Traffic Model Agent API

For proper communication between the client and traffic model, we have defined a structure for the payload itself. For the traffic input, we used the GMNS standard. See the example below:

```
"payload": [
               {
                       "id": "1",
                       "type": "link",
                       "standard":"https://github.com/zephyr-data-specs/GMNS",
                       "change-type":"attributeChange",
                       "capacity":1200,
                       "free_speed":70
               },
               {
                       "id": "23",
                       "type": "node",
                       "standard":"https://github.com/zephyr-data-specs/GMNS",
                       "change-type":"attributeChange",
                       "ctrl_type":"yield"
               }
       ]
}
```

Message Payload:

- id: unique and stable identifier within the dataset
- type: type of the entity
- standard: pointer to the (refined) schema the entity conforms to
- change-type: create | delete | attributeChange | attributeAddition | attributeRemoval
- *other variables (i.e. free speed) are defined by the GMNS standards*

We will only support changes to the links and results published based on link-based information for the beta version and beyond. However, in making this document more generic, we also provided some examples to make changes to configurations for network nodes under the GMNS standard. Similarly, one could support any other adjustments of the different object types that are given there (signals, lanes, movements ...). This should enable coverage of modifications of the whole GMNS standard and allow for easy extension of the platform if the model in question supports such modifications.

For the link based changes, the attributes that are viable for changes will be ‘capacity’ in vehicles/hour/lane, ‘free speed’ in km/h and ‘lanes’ as specified in GMNS, all integers.
Internally, both for P4ALL and KUL, this capacity per lane and the number of lanes are translated to a total link-based capacity.

Setting a capacity to 0 corresponds to a road closure. Note that this may not be viable for all roads in the network; some roads are needed to maintain connectivity (bridges, some highways...). If such a road is closed, no valid traffic state can be calculated, and the traffic model will return a result labelled erroneous (see below in the section Output Format).

There is no existing standard to be used for the traffic model output, and thus, we propose the following message structure:

```
{
   "topic":"road-network-pilsen-results",
   "clientid":"p4all-tramod",
   "payload":{
        "metadata":{
        "type":"links",
        "status":"ok/error",
        "error_msg":"",
        "simulation_time_intervals":[
               1615999314,
               1616002914,
               1616002915
        ],
        "road_network_geom_url":"www.tramod.com/pilsenmod.json",
        "units":{
               "cost":"hour",
               "flow":"vehicles per simulation_time_interval per link"
        }
        },
        "result":[
        {
               "id":125,
               "flow":[
               1256,
               1500
               ],
               "cost":[
               0.08,
               0.05
               ]
        },
        {
               "id":126,
               "flow":[
               1865,
               1500
               ],
               "cost":[
               0.06,
               0.07
               ]
        }
        ]
   }
}
```

Message Payload:

The output format accommodates both dynamic and static traffic assignment results. For now, we publish link-based results. The ids specified correspond to the links provided in the road network data. In the future, one could also provide results related to turns or nodes; however, this is out of scope for the beta version.

"Simulation_time_intervals": :$[t_1, t_2, .., t_N]$ indicate the time horizon of the model and its temporal discretization with t_n in time since epoch. If there are *N* elements in 'simulation_time_intervals', there will always be *N-1* elements in the cost and flow arrays provided for each edge, one for each interval. E.g., The first flow value represents the flow between [*t1, t2*], the second value between [*t2, t3*] and so on. The length of each of these time intervals will always be identical.

The description of the fields in the Message Payload is listed below:

- metadata: contains additional information provided by the model
    - type: refers to the type of output from the model; for now, all the results are link-based
    - status: ok / error
    - error_msg: error message
    - simulation_time_intervals: list of provided time intervals
    - road_network_geom_url: link to the model geometry
    - units: specification of the units provided by the model
- results: contains the results of the calculation it self
    - id: unique and stable identifier within the dataset
        - here a reference to the link ids
    - flow: No. of vehicles passing in specified time interval
    - cost: the time required to cross the link in specified units

4.3.2. Noise Model Agent API

Noise modelling contains many variables that can influence the calculated noise levels. The most important (in the city environment) being traffic volumes. Thus, the starting point of our noise modelling calculation is not direct input from the user but a message payload from traffic assignment. Thus, the input for noise modelling is the same as the output from traffic model agent API (as described in paragraph 4.3.1).

The output noise modelling payload will return new noise levels for predefined observation points (virtual microphones). See the example here:

```
{
   "topic":"noise-calculation-result-pilsen",
   "clientid":"noisemodelling-p4all",
   "payload":{
      "url":"http://vps17642.public.cloudvps.com:8081/data/road-network-calculation-result-
pilsen"
   },
   "data":{
      "metadata":{
         "type":"links",
         "status":"ok/error",
         "error_msg":"””",
         "receiver_geom_url":"www.tramod.com/pilsen_noise_receivers.json",
         "units":{
            "laeq":"dB at receiver point"
         }
      },
      "result":[
         {
            "id":125,
            "laeq":88
         },
         {
            "id":150,
            "laeq":90
         }
      ]
   }
}
```

4.3.3. Air Quality Emission model

The Air model will return concentrations for several compounds based on static background data and emissions derived from traffic data. Emissions are calculated from the road network properties and the traffic volumes on the road links. The input for the Air model is taken from the output of the traffic model and thus using the format of the traffic model agent API.

An example of the returned data:

```
{
    "topic": "air-calculation-result-1-pilsen",
    "clientid": "air-tno",
    "payload": {
        "url": "http://vps17642.public.cloudvps.com:8081/data/air-calculation-result-1-pilsen"
    },
    "data": {
        "type": "FeatureCollection",
        "features": [{
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [13.41168405784886, 49.7721183108128]
                },
                "properties": {
                    "C_NO2": 20.3794327
                }
            }, {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [13.390445548912535, 49.729302642721855]
                },
                "properties": {
                    "C_NO2": 26.225502
                }
            }
        ]
    }
}
```

In the ‘data’ section, the results for the calculation are GeoJSON points, each having properties. In this example, the concentration values for the NO2 compound are shown. In future versions, other compounds may be listed as well (PM10, PM25 etc.). The unit of the concentration values is ug/m3.

5. Conclusion and Future work

With this demonstrator deliverable and the Closed Beta demonstrator as a PoC, it is shown that an architecture built around the Message Broker in the DUET core is capable of creating an effective chain of models.

A scalable solution is demonstrated, using the User Interface for executing several User Stories. When a road element is obstructed by a user as a ‘what-if’ scenario, the system propagates this change to a traffic model. The traffic model calculates the impact on the road infrastructure and signals the DUET system of the data changes. These changes trigger the Noise and Air Quality models to calculate the impact of the changes for their domain. At completion, the models signal the DUET core of their data changes. The overall data changes are picked up by the User Interface for evaluation by the user.

The closed beta version of the DUET system currently requires the models to be running to execute the User Stories. Also, the separation of scenarios is done by temporarily introducing a ‘scenario id’ in the message exchange.

Future work includes the adaptation of the Model Agent API as described in D3.5. This API will be responsible for deploying models, data and scenario context on demand, such that users can execute their stories independently of each other for every independent scenario.

Future work also includes formalisation of how and where (static) data is to be stored and how Data Context Graphs are to be used to register the data changes (dynamic data). Current models will need to be expanded to support the data storage mechanisms and the Model Agent API for inclusion in the final DUET system.

6. References

- D3.2 IoT stack and API specifications v2
- D3.5 DUET Cloud design for model calibration and simulation
- D4.2 DUET Data integration
- D4.5 DUET standard reports and data analysis tools
- D5.1 DUET System architecture & Implementation plan
- D5.2 DUET Initial Digital Twin Prototype