Deliverable

D3.5 Cloud design for model calibration and simulation

Project Acronym:	DUET	
Project title:	Digital Urban European Twins	
Grant Agreement No.	870697	
Website:	www.digitalurbantwins.eu	
Version:	1.0	
Date:	30 November 2020	
Responsible Partner:	TNO	
Contributing Partners:	KUL, ATC, UWB	
Reviewers:	**Internal** Gert Vervaet (AIV) Philippe Michiels (IMEC) Karel Charvat (P4All) **External** Yannis Charalabidis	
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	25.09.2020	Max Schreuder	TNO	Initial structure
0.2	02.10.2020	Walter Lohman	TNO	Basics
0.3	05.10.2020	Walter Lohman/Hans Cornelissen	TNO	Text fixes
0.4	12.11.2020	Walter Lohman/Max Schreuder	TNO	for internal and external review
0.5	17.11.2020	Walter Lohman	TNO	processed comments from internal review
0.6	18.11.2020	Hans Cornelissen/Walter Lohman/ Max Schreuder	TNO	version ready for review
0.7	25.11.2020	Walter Lohman/ Max Schreuder	TNO	provision of final version
0.8	27.11.2020	Expert review	External experts	External expert comments processing
0.9	30.11.2020	Max Schreuder/Walter Lohman	TNO	processed external review comments
1.0	30.11.2020	Max Schreuder	TNO	Final version

Table of Contents

Executive Summary 5
1. Introduction 6
2. API Specification 8
2.1 Model API Messages 9
2.2 Model Agent API Messages 10
2.3 Data API Messages 12
2.4 Model to Model discussion 14
3. Model specific API 15
3.1 Noise model extensions 15
3.2 Air model extensions 15
3.3 Traffic model extensions 15
4. HPC in DUET 16
4.1 HPC Architectures 16
4.2 HPC for the DUET models in containers 17
4.3 HPC example Air Model 18
4.4 HPC development for the Dynamic Traffic Assignment Model 18
5. Conclusions 19
6. References 21

Tables

Table 1: Model API Messages formats 10
Table 2: Model Agent API Messages formats 12
Table 3: DATA API Messages formats 13

Figures

Figure 1: Cloud design for models 7
Figure 2: Model activation & execution sequence 8
Figure 3: DUET multi model integration 14
Figure 4: Visualisation examples of HPC Cluster/Grid and HPC GPU 17
Figure 5: HPC using containers 17
Figure 6: Parallel executing concentration calculations on HPC GPUs 18

Executive Summary

The DUET system contains multiple models to perform the calculations for the Digital Twin. The DUET T-cell architecture enables the integration of these models of the DUET system. Computational models (air, noise, traffic) are integrated in DUET by connecting to the DUET T-Cell by means of a suitable API.

This API will enable the models to be controlled by the DUET system. The API will facilitate the start and stop of the simulation, calibration and validation, the exchange of necessary data and results. The API will be accessible using a gateway to the Apache Kafka Platform[1], and will relay messages between Kafka and the models. Kafka functions as the main message streaming platform in the DUET T-Cell architecture.

The availability of the individual models is realised using Docker containers[2]. The individual models will be packaged inside a Docker container enabling deployment anywhere in the available cloud, thus forming a cloud of available models. The models will run outside the DUET T-Cell and are interconnected using the API. A Docker Orchestrator is implemented for starting up, retrieving status and termination of the individual model docker containers.

The DUET system can effectively use HPC for model calculations. Most of the models require extensive computational power and can have a long runtime; hours, days, weeks. In order to achieve a fast response to requests for 'What-If Scenario' simulations, model calculations can be divided into a lengthy pre-processing calculation/calibration stage and a fast calculation for changes of a smal(ler) scale. The pre-processing stage can use HPC to shorten runtime or models can be ported to run on HPC hardware directly.

This deliverable highlights HPC architectures available to the current models in DUET;

- HPC Cluster/Grid. Multiple servers connected by a fast network.
- HPC Cloud. Almost the same as the HPC Cluster, only the servers are available on the Internet.
- Docker services (Linux/Windows) and Service Fabrique (Microsoft) will spawn models on suitable resources in the cloud.
- Multi-Core, GPU or Multi-GPU. This refers to methods of executing models or compute intensive parts of the model in parallel using suitable hardware on one (or more) machine(s).

Primarily, this deliverable provides the generic description of the minimal API functions for model integration in the DUET architecture.

[1] https://kafka.apache.org/

[2] https://www.docker.com/

1. Introduction

This deliverable provides the generic description of the minimal API functions for the purpose of model integration in the DUET architecture. It also discusses HPC and its use for training, calibrating and simulation of the models.

The first and second chapter of this deliverable discusses the model API specifications, provides examples of API messages and model specific messages. The third chapter discusses how the API can facilitate HPC for DUET.

An overview and description of the DUET models (Air, Traffic, Noise) is given in **Deliverable D3.3 Smart City domains, models and interaction frameworks v1**. This deliverable also describes the required input and output data for these models.

This deliverable must be read within the context of the following deliverables;

- **D3.1 IoT stack and API specifications v1** describes the components onboarding of data into DUET
- **D3.3 Smart City domains, models and interaction frameworks v1** describes the models that the different partners can provide to DUET
- **D3.4 Smart City domains, models and interaction frameworks v2** (M24), will describe and report on the model adaptations for DUET
- **D3.8 Digital Twin data broker specification and Tools v1** describes the high level roadmap and technical architecture and implementation tools and frameworks of the digital twin data broker from the viewpoint of DUET
- **D5.1 System Architecture & Implementation Plan** discusses an overall architecture and implementation or deployment plan
- **D3.10 Multi Layered security model specification** discusses the separation of the Kafka Message Streaming Platform and the connection APIs to bring extra security

A central component of the DUET T-Cell is the message streaming platform, Apache Kafka, for the exchange of events and data. Communication and Data Exchange with the T-Cell and thus other parts of the DUET system is done using the Kafka platform. The models define their own ‘channels or topics’ for exchanging information with the system.

The data to be exchanged with the model or the Docker Orchestrator will be formalized in a minimal required API. Channel definitions and additions to the API for specific model interactions will be described in the relevant model sections. An overview of this model cloud is given in figure 1.

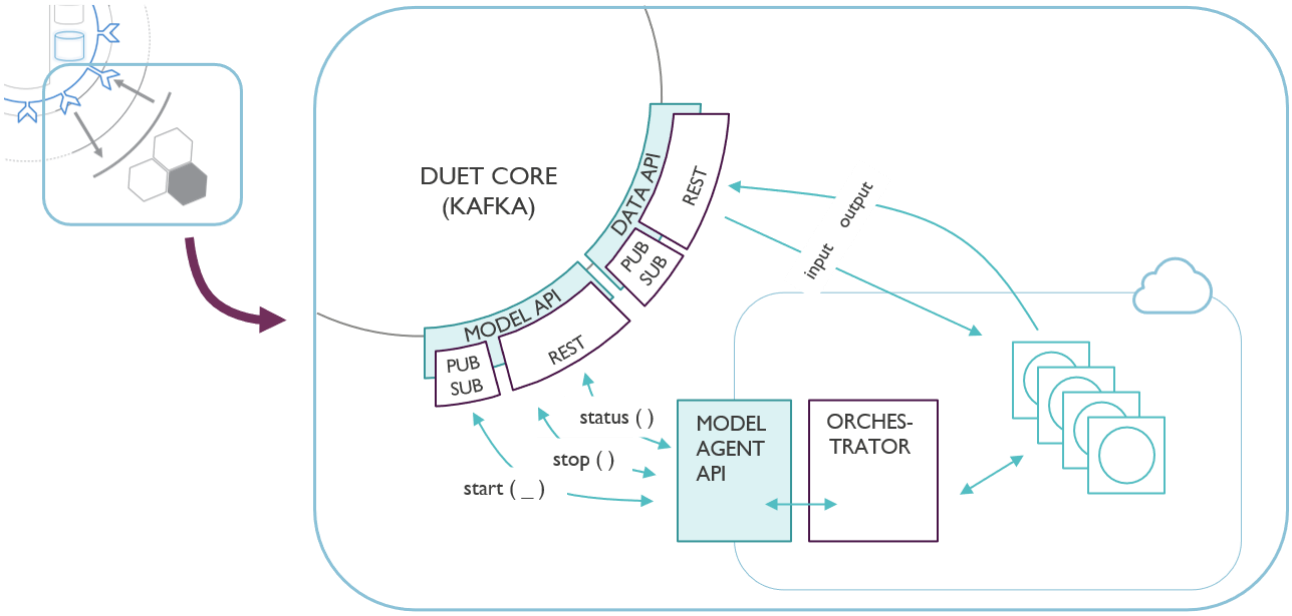


Figure 1: Cloud design for models

The models are packaged inside a Docker Container, making deployment in the cloud possible. Where, When and Which models are deployed when requested will be controlled by a Docker Orchestrator such as Kubernetes[3]. A set of suitable scripts will regulate the interaction with the DUET system and the Models in the Cloud.

[3] https://www.docker.com/products/kubernetes

2. API Specification

The models in the cloud are controlled by a Model Agent and corresponding API and deployed upon request. An Orchestrator is responsible for deployment of models, taking platform and resource requirements into account. The Docker technology will use a set of scripts based on Kubernetes[4] or similar. The Orchestrator receives messages indirectly through the Model Agent API, and executes the requests to Start (Deploy), Stop (Delete) the models.

The start request received from the DUET system will include a Model context (parameters, data, …) to be pushed and passed to the model. Response messages from the model include a unique instance Id to optionally control or get information from the specific model.

When a model is deployed, it should listen for command messages from the Orchestrator/Model Agent API and handle data through the DATA API. Necessary properties, data or links to data sources can be passed in the context section included with the startModel request.

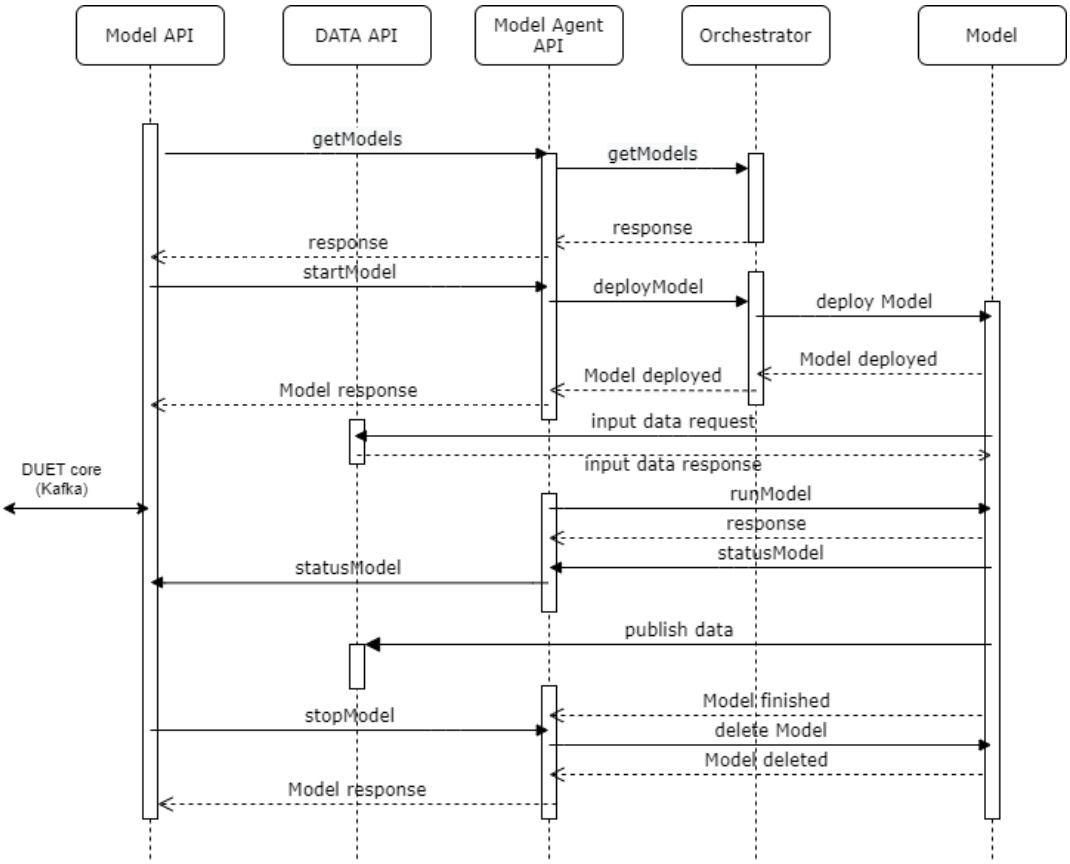


Figure 2: Model activation & execution sequence

The DUET T-Cell has two major RESTful APIs; The Model API and the Data API. RESTful APIs provide a standardised way of communicating with data sources on the web. REST can be implemented with a multitude of technologies but we are focusing specifically on RESTful HTTP here.

[4] https://kubernetes.io/

Although the "verbs" of the HTTP protocol are fixed (GET, POST, PUT, PATCH, DELETE,…) and known, their implementation can differ from API to API. Many best practices exist and the use of REST frameworks to build APIs can mean a standardised way of building the API but it is not enforced (you can always add your own endpoints that misuse the HTTP verbs). HTTP response codes are similarly untrustworthy since their correct usage is also not enforced[5] . Lastly, any HTTP method can return data on a request (even though it's not the correct RESTful implementation) whether it's in the body

Similarly, HTTP headers are not always used correctly or consistently across API's (or even within the same API). Simple things like the letter case a header is written in can mess up case-sensitive REST libraries and incorrect header values (by using existing headers for proprietary information) are allowed but destructive.

In the following section we concentrate on the messages that need to be exchanged through the API's. We've chosen to use the JSON message format, since its standard is well known and widely used.

2.1 Model API Messages

When connecting the Models to the T-Cell, a Model Agent API defines the methods necessary to use a model from the DUET system. Models reside somewhere in the cloud and need to be deployed and run with a proper context when they are requested from the DUET system. When a model is deployed, a unique ID will be assigned to the deployed instance of the model. The Model data exchange will be done using the Data API.

Model API Messages formats		
Request/ response	**Description**	**JSON example**

generic format	This is the general format of all Model Agent API JSON commands	`{`
		`  "cmd": "<Model-Agent-API-Command>",`
		`  "payload": {}`
		`}`

[REQUEST] getModels	Request a list of known available models	`{`
		`  "cmd": "getModels",`
		`  "payload": {}`
		`}`

[RESPONSE] getModels	List of available models	`[`
		`  {`
		`  "modelname":"<model-name>",`
		`  "status": "<model-status-enum>"`
		`  }`
		`]`
		
		`"<model-status-enum>":`
		`[ "ready",`
		`  "initializing",`
		`  "running",`

[5] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Model API Messages formats		
Request/ response	**Description**	**JSON example**
		"calibrating", "stopping", "finished", "error", "unknown"]

[REQUEST] startModel	The startModel request will include the information of the model to be started, together with the context to be passed to the model. The context should include all information the model needs to run like inputs, outputs, extent, time bounds etc.	{
		"modelname":"<model-name>",
		"context":
		{
		"property_1": "....",
		"property_2": "....",
		"property_n": "....",
		"input_data_url": "https://my.data",
		"output_data_url": "https://my.data"
		}
		}

[RESPONSE] startModel		{
		"id": "<model-instance-id>",
		"status": "<model-status-enum>"
		}

[REQUEST] stopModel	The stopModel request will pass the id of the model to be stopped.	{
		"id": "<model-instance-id>"
		}

[RESPONSE] stopModel		{
		"id": "<model-instance-id>",
		"status": "<model-status-enum>"
		}

[PUSH] statusModel	This message is sent to the Model API without a request; initiated by the model when it's status changes.	{
		"id": "<model-instance-id>",
		"status": "<model-status-enum>"
		}

Table 1: Model API Messages formats

2.2 Model Agent API Messages

When a request from the DUET system for a model is received, the model image should be located. Currently this is done using an Orchestrator, which has a register of deployable models with their resource and platform requirements. At a later stage of the DUET development this can also be fulfilled using a Model Catalog inside the DUET core.

The requested model will be deployed, the necessary context is exchanged and a unique instance ID is

assigned. This ID allows control and information exchange with the model.

The Model should listen for the messages from the Model Agent (or Orchestrator). The minimum set of API messages the Model should respond to are listed in this section.

Model Agent API Messages formats		
Request/ response	**Description**	**JSON example**
generic format	This is the general format of all Model JSON commands	`{ "cmd": "<Model-Command>", "payload": {} }`
[REQUEST] deployModel	The deployModel request will include the information of the model to be started, together with the context to be passed to the model. The context should include all information the model needs to run like inputs, outputs, extent, time bounds etc.s	`{ "cmd": "deployModel", "modelname":"<model-name>", "context": { "property_1": "....", "property_2": "....", "property_n": "....", "input_data_url": "https://my.data", "output_data_url": "https://my.data" } }`
[RESPONSE] deployModel		`{ "id": "<model-instance-id>", "status": "<model-status-enum>" }`

[REQUEST] runModel	The runModel request will start the deployed model in the correct mode.	`{ "cmd": "runModel", "id": "<model-instance-id>", "mode": "<model-mode-enum>" }`
		
		`"model-mode-enum": [ "calibrate", "calculate", "validate" ]`

[RESPONSE] runModel		`{ "id": "<model-instance-id>", "status": "<model-status-enum>" }`

Model Agent API Messages formats		
Request/ response	**Description**	**JSON example**
[REQUEST] deleteModel	The deleteModel request will pass the id of the model to be stopped and undeployed.	`{ "cmd": "deleteModel", "id": "<model-instance-id>" }`
[RESPONSE] deleteModel		`{ "id": "<model-instance-id>", "status": "<model-status-enum>" }`
[PUSH] statusModel	This message is pushed by the Model without a request; initiated by the model when its status is updated.	`{ "id": "<model-instance-id>", "status": "<model-status-enum>" }`

Table 2: Model Agent API Messages formats

2.3 Data API Messages

The Data API of the DUET core is used to exchange the data to and from the models. At the start request for a model, a suitable context with data or links to data sources is included. When the model is deployed, it uses this information to fetch the data needed for calculation, calibration or validation processes.

Changes of data need to trigger events in the DUET core. The Kafka Message Steaming platform supports this using a Publish/Subscribe mechanism. The Kafka Streaming platform is not exposed outside the DUET core and the Data API and corresponding Message Gateway will relay or trigger corresponding messages to Kafka.

A choice still has to be made if multiple data sets share one topic or the payload contains an ID to separate those.

The message exchange between the Message Gateway/Data API and the Model is listed in this section.

DATA API Messages formats		
Request/ response	**Description**	**JSON example**

generic format	This is the general format of all Data JSON commands	`{ "topic": "<Topic-Identification>", "payload": {} }`
		
		`<Topic-Identification>:`
		`[`

DATA API Messages formats		
Request/ response	**Description**	**JSON example**
		model.traffic.kul-data.1, model.traffic.p4a-data.1, model.air-quality.1, model.noise-pollutions.1]

[GET] /request	The Model initiates a request for input data using the URL included in the context. Not implemented yet.	{
		"topic": "<Topic-Identification>"
		}

[PUSH] /request	This is not yet clear, when DUET has new data for the model. The Core should push this data to the model.	{
		"topic": "<Topic-Identification>",
		"payload": {}
		}

[POST] /publish	When a Model has data, it pushes this through the Data Gateway.	{
		"topic": "<Topic-Identification>",
		"payload": {}
		}
		
		"payload" containing data to be pushed.

[POST] /publish	When data change occurs somewhere in the DUET system. Models should receive the new data. This data is PUSHed towards the Model.	{
		"topic": "<Topic-Identification>",
		"payload": {}
		}

Table 3: DATA API Messages formats

The API for data exchange is work in progress. The current API is described here: http://20.67.144.95:8081/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config .

2.4 Model to Model discussion

The DUET system is an integral system containing multiple components that are dependent on each other. An Air Quality model, for example, relies on the output of a Traffic Model. It should recalculate the Air Quality if there is a change in the calculated Traffic Data.
The DUET system uses the Publish/Subscribe mechanism to interconnect these dependent Models. As soon as a Traffic model publishes new outputs, a corresponding event is triggered on the Kafka Message Streaming Platform thus notifying Data gateways. Data gateways will relay the data to 'Subscribers' that are subscribed to the 'Topic' in the published data.

This mechanism indirectly takes care of Model to Model data exchange. When Models need to exchange data, there has to be an agreement on the published or subscribed 'Topic'. This topic is still a point of discussion and may be modified at a later stage in the DUET development.

Figure 3 shows a sequence diagram for a dependent model run. First a Traffic model is run which updates information of the traffic volumes. Next the Noise model is run, so changes to the traffic volume are reflected in the calculated Noise Pollution. This sequence diagram (Figure 3) shows there is no direct connection between the models. The interconnection is accomplished using the Message Streaming Platform.

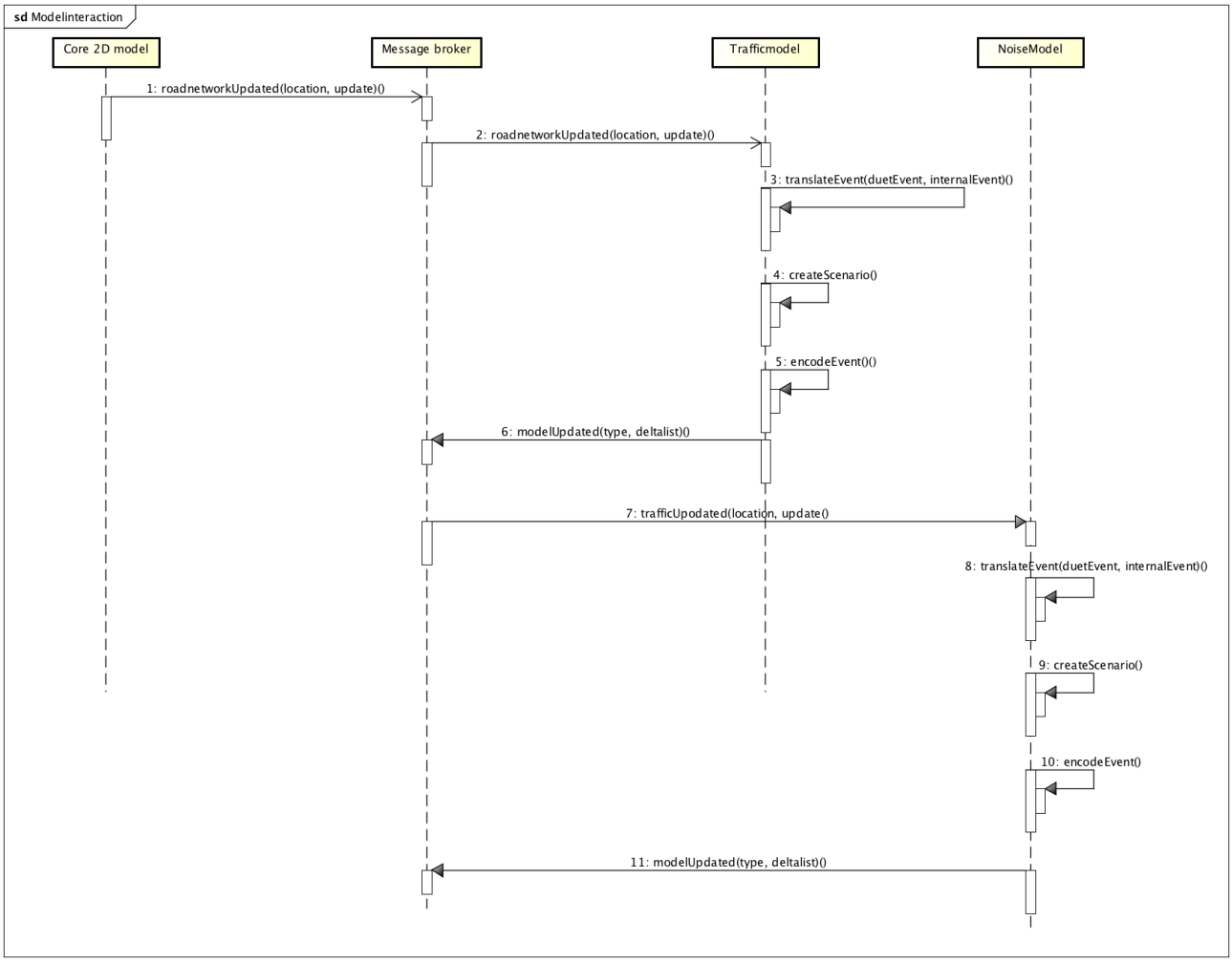


Figure 3: DUET multi model integration

3. Model specific API

The individual models can support an extended API. These extensions could be described here when they become available.

3.1 Noise model extensions

For now, there are no model specific extensions to the messages. This might change in a future stage in the development of the DUET platform.

3.2 Air model extensions

For now, there are no model specific extensions to the messages. This might change in a future stage in the development of the DUET platform.

3.3 Traffic model extensions

For now, there are no model specific extensions to the messages. This might change in a future stage in the development of the DUET platform.

4. HPC in DUET

The DUET system contains multiple models to perform calculations for the Digital Twin. Most of the models require extensive computational power and can have a long runtime; hours, days, weeks. In order to achieve a fast response to requests for 'What-If Scenario' simulations, model calculations can be divided into a lengthy pre-processing calculation/calibration stage and a fast calculation for changes of a smal(ler) scale.

HPC architectures help us to execute these pre-processing calculations within an acceptable runtime. Using the results from the pre-processing step, it is possible to achieve a fast response to deviations initiated by the DUET system user. Also long training and calibration of models can effectively be done using HPC. DUET can then have fast feed forward models relying on this training/calibration for forecasting and prediction.

Another method of using HPC is to enable models to use HPC architectures directly. The use of GPUs to perform calculations in parallel can gain enough speed for direct use in the DUET system. Models most likely will not run out-of-the-box on these highly parallel driven architectures, but need to be ported/converted to benefit.

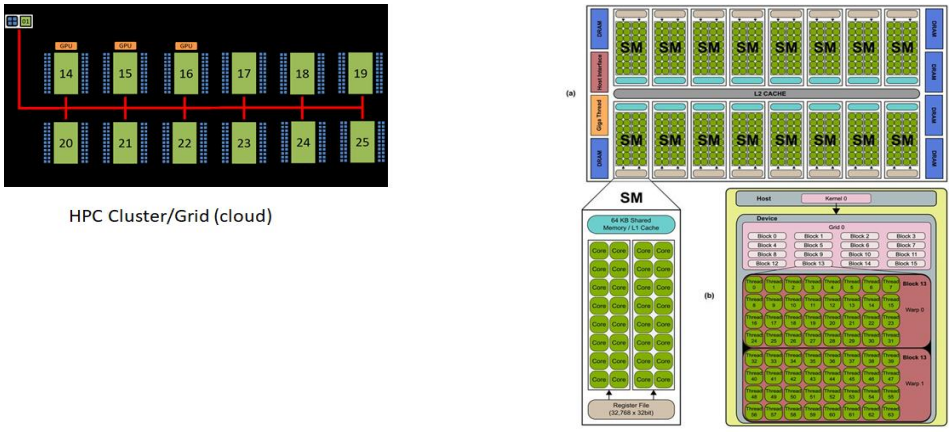
DUET's modular architecture will be an enabler to effectively use HPC architectures. This chapter will describe relevant HPC architectures for the current models in DUET.

4.1 HPC Architectures

Currently there are multiple architectures that can be referred to as HPC architectures. Suitable HPC architectures for the DUET models are the following:

- HPC Cluster/Grid. Multiple servers connected by a fast network. Computations can be divided among the available servers in the cluster. Mostly a batch oriented approach, suitable for lengthy pre-processing calculations, training and calibration.
- HPC Cloud. Almost the same as the HPC Cluster, only the servers are available on the Internet. Especially if a training or calibration step only needs to be calculated once, it can be executed on an Internet Cloud, instead of having an HPC Cluster on premise.
- Docker services (Linux/Windows) and Service Fabric (Microsoft) will spawn models on suitable resources in the cloud. Computational power can be dynamically increased or decreased, according to the need.
- Multi-Core, GPU or Multi-GPU. This refers to methods of executing models or compute intensive parts of the model in parallel using suitable hardware on one (or more) machine(s). See ITEA3 - MACH[6].

[6] https://itea3.org/project/mach.html



HPC Cluster/Grid (cloud)

HPC GPU architecture (Nvidia)

Figure 4: Visualisation examples of HPC Cluster/Grid and HPC GPU[7]

4.2 HPC for the DUET models in containers

In DUET, models for multiple domains are required to perform the use-case calculations. HPC clusters can be used to run the individual models across different resources in the cloud. A container with a model can run anywhere in the cloud while still working together with the other models by using the DUET T-Cell architecture. With this method the necessary computational power for the set of models is divided over the available resources in the HPC cluster/grid.

The flexible DUET Publish/Subscribe architecture based on the Kafka Message Streaming Platform can be used to determine if a (new) pre-processing step is necessary. By defining the dependency of a model for pre-processed results using an assigned 'Topic', the system can deploy the pre-processing model to a HPC instance when pre-processed data for the fast model is not yet present. When ready, the pre-processing model publishes the results back to the DUET system and thus enables DUET to use the fast model.

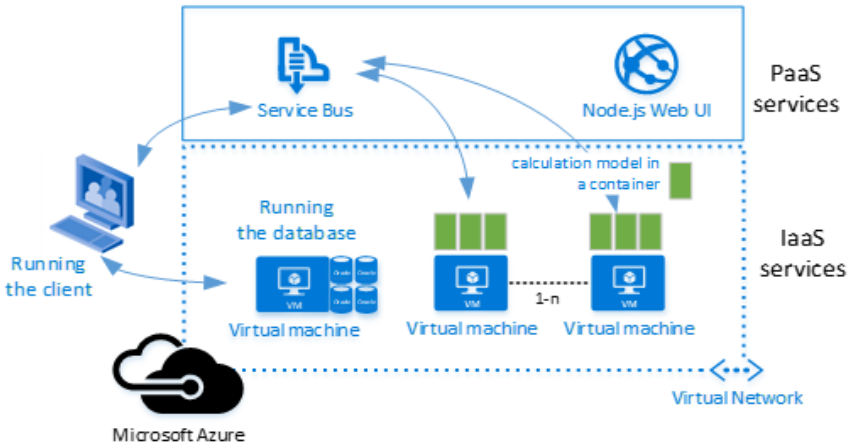


Figure 5: HPC using containers[8]

[7] source: https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs

[8] source: Microsoft

4.3 HPC example Air Model

The Air Model is an example of a model that can utilize HPC within the DUET architecture. The Air Model calculations consist of multiple stages, see also figure 6;

1. **Pre-processing stage;** calculating background concentrations based on emissions from emission databases. Calculations are done using an HPC cluster with 12 servers containing 64 CPU cores. The calculations run in 3 stages from coarse to fine. The first stage calculates for a 30km x 30km, scale of Europe. The Second stage is a 7km x 7km on country scale and the last stage is a 1km x 1km resolution at city scale. These calculations are scaled across the HPC cluster using OpenMP in a batch oriented approach.
2. **Interactive stage;** calculating contribution of traffic emissions to the background from the pre-processing stage. Concentrations are calculated for receiver points independently, executed in parallel on NVidia GPU hardware.

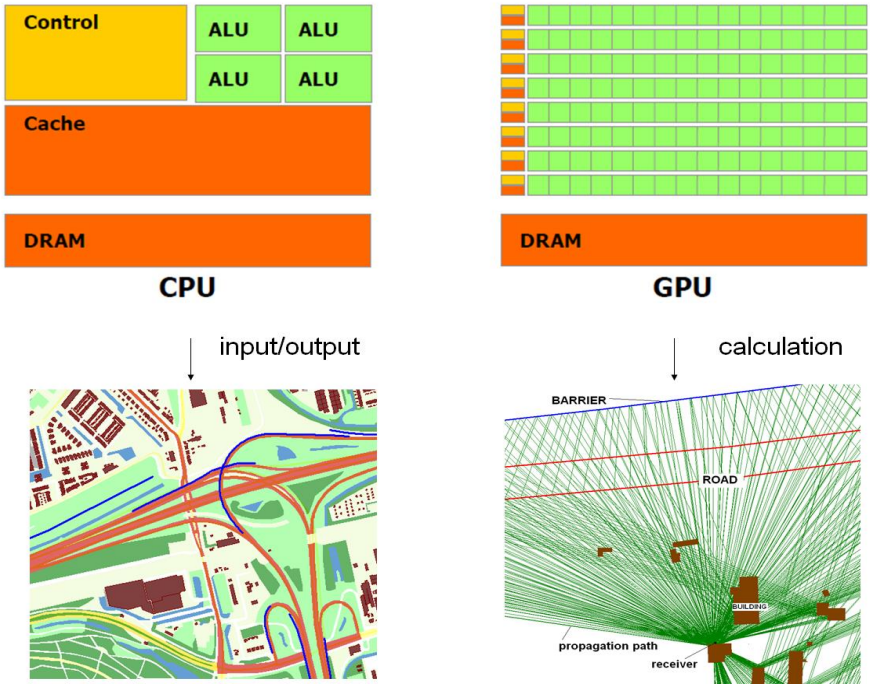


Figure 6: Parallel executing concentration calculations on HPC GPUs[9]

4.4 HPC development for the Dynamic Traffic Assignment Model

Deliverable D3.3 Smart City domains, models and interaction frameworks v1, details some avenues for research on HPC use for traffic models and highlights some of the restrictions we're facing in the use of supercomputers (see page 16-17). The methodology being used here is still under development and will be elaborated on in Deliverable D3.4 Smart City domains, models and interaction frameworks v2.

[9] source: TNO Poster 2011 for ISC: Urban noise mapping using parallel computation on a Graphics Processing Unit (GPU) (www.isc.org)

5. Conclusions

The DUET T-Cell architecture and its modular approach, enables dynamic and on-demand attachment of models to the DUET system. The use of a message streaming platform also allows the DUET system to NOT set a fixed sequenced chain of models. Models will be signalled to run, when changed data or event messages instruct a model to perform a 'run'. This topic is still a point of discussion and may be modified at a later stage in the DUET development. Other open discussions are listed below.

Further discussion relates to the 'deleteModel' request listed in Chapter 2.2. It is not yet clear if this is a necessary or wanted function.

> *"Alternative to 'deleteModel' could be a counter of connections to a model and let it get deleted when that count is 0. Or will there always be exactly one visual or other client connected to a model. Then we could have a start (starts if not available, connects to existing when it is) and release or stop (decrease counter and delete when counter is 0). Some models may need to 'stay alive' and will be used for different whatifs."*

> *"This is hard to do if not all models are reentrant. From docker it is easier to start the container on request, but there can be multiple independent instances of the model. Sharing of a started model might be possible, however it is not yet clear how this can be achieved easily. For example, TNO's Air Model calculates for a specific region. The supplied context at start points to the necessary data and after retrieval the model starts calculating. Currently it won't know what to do when a new context/start is received. When looking from a container perspective, for a new start a new container with the model is started independent and supplied with the context of the new request. This model possibly will be deployed somewhere else. Maybe if models exit themselves when ready and are undeployed upon exit, the deleteModel is not necessary."*

Further discussion is required on the DATA API described in Chapter 2.3, this discussion relates to Topics vs Datasource references;

> *"Topics should be opaque, as Users should address data sources, and not topics."*

> *"However, when pushing data from the model it should have a reference to the topic. This is necessary to create appropriate events for data change signaling."*

At a later stage of the DUET system development, these open points must be addressed.

HPC using (Docker) Containers for the models, creates a manageable method for attaching models to the DUET system on-demand and have these models run on platforms with the ideal resources to run effectively. DUET's modular architecture can be an enabler to effectively use HPC architectures for the Digital Twin.

At this stage models are not yet adapted for the DUET system and HPC. Although this deliverable describes the possible methods of adaptation, the actual adaptation of the models is still work in progress. At this time there is limited data available of how models can benefit from HPC. For the Air Quality model (TNO) the port to an HPC enabled model decreased calculation times by approximate 200 times (3 hours to 1 minute). Pre-processing steps are already implemented on an HPC cluster, but not modified for use in the DUET system. For this model and the other DUET models the adaptation to HPC will be further researched and reported in **Deliverable D3.4 Smart City domains, models and interaction frameworks v2**, which is scheduled for M24.

Although not elaborated within this Deliverable, the separation of the message bus (Kafka) of the DUET T-Cell and the Model Agent API, could bring an extra level of security into the system. The use of a well-defined API for models to connect, prevent unsolicited messages from entering the heart of the DUET system, the T-Cell architecture. For a detailed discussion of security matters, we refer to **Deliverable D3.10 Multi Layered security model specification**.

6. References

- DUET - Digital European Urban Twins https://www.digitalurbantwins.com/
- DUET Technical Architecture document https://docs.google.com/document/d/1DO5PB22wCzig1mlvoG_LYNA9ebEtmr4lV2uek9C6hZA/edit?usp=sharing

- Deliverable D3.3 Smart City domains, models and interaction frameworks v1
- Deliverable D3.4 Smart City domains, models and interaction frameworks v2
- Deliverable D3.8 Digital Twin data broker specification and Tools v1
- Deliverable D3.10 Multi Layered security model specification
- Deliverable D5.1 System Architecture & Implementation Plan

- ITEA3 - Massive Calculations on Hybrid Platforms. https://itea3.org/project/mach.html
- Lotos Euros - AIR QUALITY MODELLING AND EMISSIONS. https://lotos-euros.tno.nl/
- Kubernetes - open-source system for automating deployment, scaling, and management of containerized applications https://kubernetes.io/