Towards a Digital Urban Climate Twin: Simulation-as-a-Service (SaaS) for Model Integration

Author(s):
Aydt, Heiko

Publication Date:
2020-11-11

Permanent Link:
https://doi.org/10.3929/ethz-b-000455866 →

DELIVERABLE

TECHNICAL REPORT

Version 11/11/2020

D3.2 – Towards a Digital Urban Climate Twin: Simulation-as-a-Service (SaaS) for Model Integration

Project ID	NRF2019VSG-UCD-001
Project Title	Cooling Singapore 1.5: Virtual Singapore Urban Climate Design
Deliverable ID	D3.2 – Towards a Digital Urban Climate Twin: Simulation-as-a-Service (SaaS) for Model Integration
Authors	Heiko Aydt
DOI (ETH Collection)	
Date of Report	11/11/2020

Version	Date	Modifications	Reviewed by
1	29/09/2020	Version 1	Jimeno Fonseca

1 Abstract

A Digital Urban Climate Twin (DUCT) is a set of specialised urban climate models and anthropogenic heat emission models needed to closely resemble the urban climate dynamics for a particular city of interest. It allows its users to experiment with a digital representation of the city, its urban climate and the various anthropogenic contributors to urban heat (e.g., industry, traffic, and buildings). The outcomes of such an experimentation can be used to support research as well as planning and decision making. Conducting simulation-based experiments and what-if analyses requires to carry out a number of use-case specific workflows involving many individual steps. These workflows also involve a variety of computational models. In order to facilitate interoperability between these models, a DUCT should be designed as a federation of loosely coupled models. In addition, the underlying middleware, that facilitates the federation of models, should be designed so its components can be deployed and operated in a distributed manner. This report introduces the Simulation-as-a-Service (SaaS) concept and discusses a SaaS middleware prototype as the foundation for the development of a DUCT for Singapore. Two case studies are discussed. The first demonstrates automated workflow execution. The second showcases the interoperability between SaaS components and an end-user decision support application. Based on our evaluation with the prototype as well as the demonstrators, we conclude that the SaaS concept presented here is suitable for realising a full-scale DUCT, including a federation of models and end-user applications. Future work will be concerned with extending the original SaaS concept further with features currently absent in the prototype. For example, this will include a dedicated data object repository as well as significantly enhanced operational security.

Table of Content

2 Introduction

Cooling Singapore uses a variety of computational tools to simulate and analyse the Urban Heat Island (UHI) effect and Outdoor Thermal Comfort (OTC) in Singapore. These analyses can cover the entire city or parts of it. Conducting simulation-based experiments and what-if analyses typically requires to carry out a number of use-case specific workflows, consisting of a variety of steps which can range from tasks such as specifying the land-use for an area of interest, converting a three-dimensional geometric model of the city, executing simulations or post-processing simulation results, for instance.

In Cooling Singapore, we broadly distinguish between mesoscale and microscale workflows. The workflows, as well as the tools and models they involve, have been analysed by (Li and Aydt, 2020). Manual execution of workflows typically requires the support of domain experts for each of the models under consideration. This is not only time-consuming and possibly prone to human error, it also limits the number of studies that can be carried out. In addition, manual execution imposes constraints on the ability to evaluate complex what-if scenarios important for researchers, urban planners and policy makers.

The concept of a Digital Urban Climate Twin (DUCT) is proposed as a tool for analysing what-if scenarios concerned with the urban climate of a particular city of interest. A DUCT is a set of specialised urban climate models and anthropogenic heat emission models needed to closely resemble the urban climate dynamics for a particular city of interest. It would allow its users to experiment with a digital representation of the city, its urban climate and the various anthropogenic contributors to urban heat (e.g., industry, traffic, and buildings). The outcomes of such experimentation can be used to support research as well as planning and decision making processes. One technical key feature of a DUCT is the capability for automated execution of complex workflows for what-if scenario analysis.

This report introduces the Simulation-as-a-Service (SaaS) concept and discusses a SaaS middleware prototype as the foundation for the development of a DUCT. The purpose of this prototype is to better understand the requirements for building a DUCT and to test suitable technologies. The SaaS middleware prototype has been developed in three phases:

1) The first phase of the prototype was used to gain a better understanding of how web service technology can be used to build interfaces for existing simulation tools that are commonly used by Cooling Singapore (e.g., WRF).

2) The second phase of the prototype was mainly concerned with the composition of multiple steps (each performed by a different SaaS component) into workflows and the practicalities of executing workflows in an automated manner.

3) The third and final phase of the prototype was concerned with further improving workflow execution capabilities as well as distributed deployment and interfacing with front-end applications.

In addition to the SaaS middleware prototype, two case studies have been implemented to demonstrate the automated execution of the workflows and the interoperability of the middleware with a front-end application. The phases of the SaaS middleware prototype provided new insights needed to build a robust SaaS middleware that can form the back-bone of a DUCT.

3 Digital Urban Climate Twin Design

Analysing the urban climate, as well as the potential effects of planned urban developments on the urban climate, requires a climate model with explicit representation of all the relevant components that influence the urban climate. In particular, this includes the significant contributors to anthropogenic heat emissions in the city. In the case of Singapore, significant contributors of anthropogenic heat include power plants, industry, buildings and road-based transport (Kayanan et al., 2019). Simulating heat emissions for each of these contributors typically require dedicated models such as power plant models (Kayanan et al., 2020), traffic models (Ivanchev and Fonseca, 2020) among others.

Explicit representation of the main contributors of anthropogenic heat as model components is important. It is these components that can be modified to create what-if scenarios. For example, in order to investigate what would happen in case that all legacy vehicles are replaced by electric vehicles, there needs to be an explicit representation of road transport and its resulting heat emissions in the DUCT. Otherwise it would be difficult to adequately describe and simulate such an all-electric what-if scenario. An important requirement for a DUCT is thus the ability to incorporate components for all relevant contributors to urban heat.

3.1 Federation of Models vs Monolithic Model

Building a single, integrated model that incorporates an urban climate model component with the various model components for anthropogenic heat emissions would be very difficult to realise due to the technical sophistication and specialisation of each of the individual model components involved. For example, while climate simulations may be based on computational fluid dynamic models, other components (e.g., traffic, buildings) may be based on agent-based, time-stepped or discrete-event models. A monolithic model that integrates such a variety of model components - so as far as it is possible in the first place - would be costly to build and maintain, not very flexible and ultimately not desirable.

A better approach would be to use dedicated model components as part of a federation of models. Each model component participating in such a federation would be responsible to simulate a specific aspect (e.g., heat emissions by traffic or building air conditioning) that is important for the overall scenario. The advantage of such a federated approach is that each of the individual model components can be developed and maintained separately by a respective group of domain experts, thus greatly reducing the complexity of a monolithic model. The primary components of a DUCT are shown in Figure 1. These components include urban climate models as well as a number of models that estimate anthropogenic heat emissions from industry, power plants, buildings, and road transport. Behind each of these components, there are sophisticated models and tools from their respective domains.

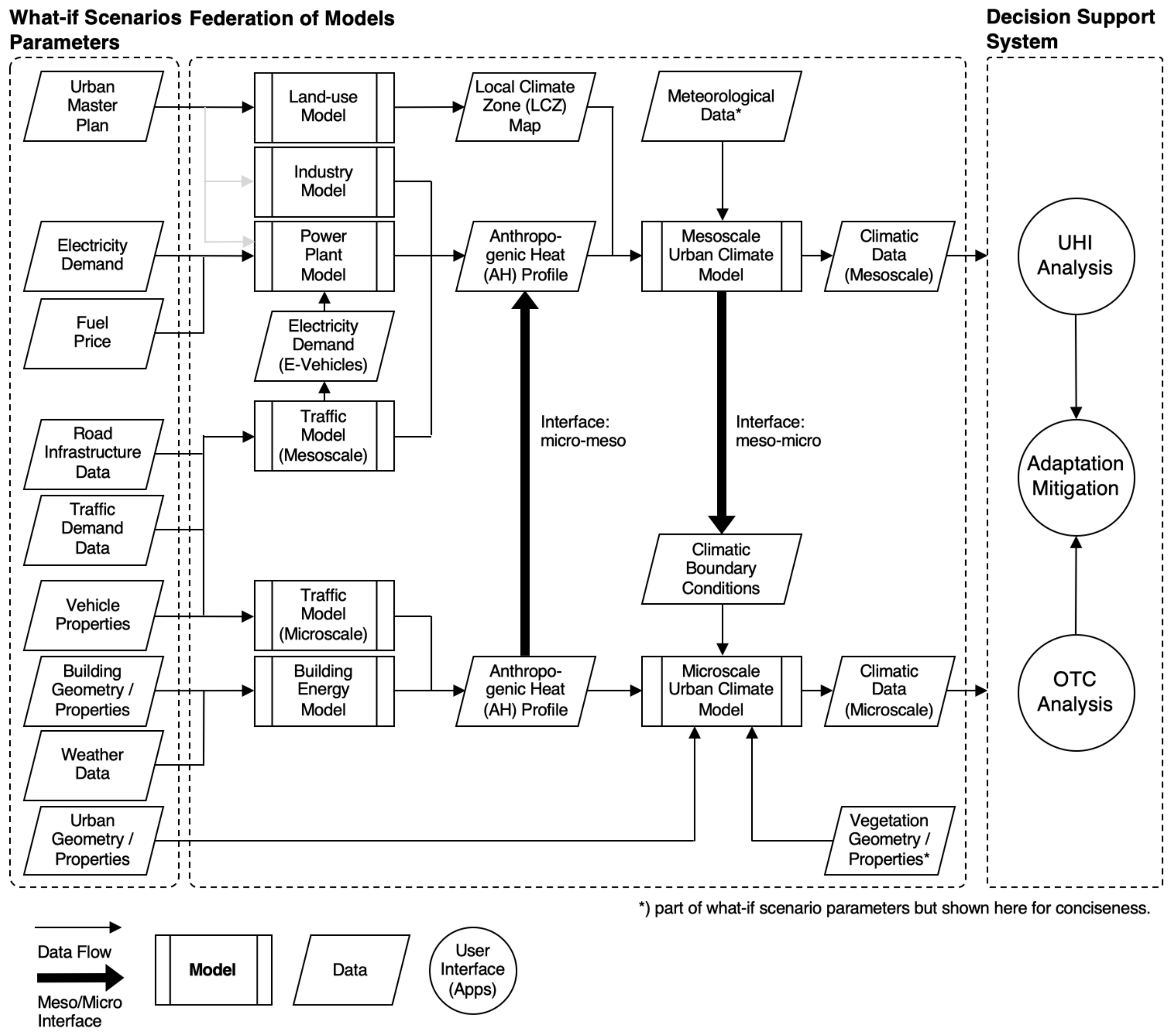
What-if Scenarios Parameters
Federation of Models
Decision Support System
Urban Master Plan
Electricity Demand
Fuel Price
Road Infrastructure Data
Traffic Demand Data
Vehicle Properties
Building Geometry / Properties
Weather Data
Urban Geometry / Properties
Land-use Model
Industry Model
Power Plant Model
Electricity Demand (E-Vehicles)
Traffic Model (Mesoscale)
Traffic Model (Microscale)
Building Energy Model
Local Climate Zone (LCZ) Map
Anthropo-genic Heat (AH) Profile
Interface: micro-meso
Anthropo-genic Heat (AH) Profile
Meteorological Data*
Mesoscale Urban Climate Model
Interface: meso-micro
Climatic Boundary Conditions
Microscale Urban Climate Model
Climatic Data (Mesoscale)
Climatic Data (Microscale)
Vegetation Geometry / Properties*
UHI Analysis
Adaptation Mitigation
OTC Analysis
*) part of what-if scenario parameters but shown here for conciseness.
Data Flow
Meso/Micro Interface
Model
Data
User Interface (Apps)

Figure 1: Overview of the principal components of a federation of models for a Digital Urban Climate Twin (DUCT).

It is important to highlight that a DUCT needs to incorporate models for all significant anthropogenic heat emitters. The various anthropogenic heat emission models shown in Figure 1 are thus indicative only.

3.2 Loose Coupling vs Tight Coupling

Simulating a what-if scenario requires the various model components of the federation to work together in order to produce the desired outcome. In other words, the model components need to interact and exchange information. In principle, there are two methods for coupling models: Tight Coupling and Loose Coupling.

Tight Coupling refers to the case where two or more model components are executed in a synchronised manner. This requires all model components in a simulation to communicate and synchronise with each other during runtime. For this purpose, a middleware is needed that facilitates synchronisation and exchange of information. The High Level Architecture (IEEE Standard 1516-2010) is a standard for distributed simulations that addresses these requirements.

Loose Coupling, on the other hand, refers to the case where the output of one model is used as input for another model without the need for models to be synchronised during runtime. It does not have the same stringent requirements as is the case for Tight Coupling. Instead, one only needs to ensure that the output of one or more model components is suitable for being used as the input for another model component. Loose Coupling possibly requires some data pre/post processing to ensure compatibility.

Since it is technically more challenging, Tight Coupling should only be used if needed in cases where two or more model components are mutually dependent on each other's output. In the case of a DUCT, there are primarily two aspects that affect the urban climate: the land-use (which largely affects how much of the incoming solar radiation is absorbed by the urban fabric and thus contributing to heat build-up) and the heat emissions by a variety of anthropogenic contributors (e.g., industry, traffic, and buildings). In this context, the flow of data is primarily unidirectional, i.e., from individual anthropogenic heat emission models to the urban climate model (see Figure 1). Loose Coupling is thus sufficient for the purpose of a DUCT.

If required, Tight Coupling can be used at the component-level. This might be the case for microscale urban climate modelling where Tight Coupling of two simulations (e.g., ANSYS

Fluent[1] and ENVI-met[2]) may be needed/desirable. Furthermore, in the case of air conditioning, there is a feedback loop which could require bidirectional flow of data. How these cases are addressed is outside the scope of this report. However, it should be noted that the general use of Loose Coupling does not preclude the possibility of using Tight Coupling for special cases.

3.3 Distributed vs Centralised Simulation

Centralised Simulation refers to the case where all model components would be executed by a single organisation within a single administrative domain. A Distributed Simulation refers to the case where individual model components are deployed and operated by different organisations in different administrative domains. Due to their integrated nature, monolithic models cannot be distributed, and thus, they can only be operated in a centralised manner.

Distributed Simulation has the advantage that owners retain control over their models as well as the data and information needed to build and operate these models. This is an important advantage when sensitive data is needed to build and/or execute a model while the model output itself can potentially be sufficiently desensitised so it can be shared. In general, Distributed Simulations allow for greater flexibility and more granular control over models and data.

3.4 Design Requirements

Given the above considerations, a DUCT should be designed as a federation of loosely coupled models. In addition, the underlying middleware that facilitates the federation of models, should be designed to support Distributed Simulation.

[1] https://www.ansys.com/products/fluids/ansys-fluent
[2] https://www.envi-met.com

4 Simulation-as-a-Service (SaaS) Middleware

Middleware is software that provides services to enable the various components of a system to communicate and manage data. In the context of a DUCT, a middleware is needed that provides services for the various model components to communicate with each other and exchange data. To this end, the Simulation-as-a-Service (SaaS) concept is proposed. The core idea of SaaS is to consider each federate (i.e., a component in the federation of models) as a service that performs computational tasks (e.g., simulations) upon request and given some input data. The outcomes of a task (i.e., output data) are made available to other federates upon completion of the task.

SaaS emphasises the need to establish common interfaces and data specifications so the exchange of data and consumption of services is facilitated. In practical terms this means that a given model component has to be wrapped by a standardised Application Programming Interface (API) that accepts requests and produces data objects in a standardised format. Furthermore, in order to enable a distributed federation of models, communication between federates should be facilitated by using established technologies, such as standard internet protocols (e.g., HTTP).

The purpose of a middleware for SaaS would be to facilitate Loose Coupling - analogue to the High Level Architecture standard for Tight Coupling. SaaS middleware can be used to build a DUCT by equipping each component of a workflow with corresponding SaaS interfaces. Each of these components can be developed, maintained and deployed by experts from a particular domain. For example, the Cooling Singapore project uses mostly existing tools (e.g., WRF[3], ANSYS Fluent, ENVI-met, City Energy Analyst[4], CityMoS[5]) to build the various model components that are needed for a DUCT.

In the remainder of this section, SaaS is introduced in greater detail.

[3] https://github.com/wrf-model/WRF
[4] https://cityenergyanalyst.com
[5] https://www.tum-create.edu.sg/research/project/citymos-city-mobility-simulator

4.1 Preliminaries

SaaS is based on the abstract notion of a process that takes some input and produces some output. Each federate will be realised as one or more of such processes. Workflows are realised by executing a number of processes in a specific sequence, essentially forming higher-level, composite processes.

4.1.1 Basic Elements

The basic elements of a federation of models have been introduced and explained elsewhere (Li and Aydt, 2020). For convenience, the introduction to the basic elements is reproduced here in verbatim with a few omissions and modifications. Figure 2 illustrates the three basic elements of SaaS: processor, function and data object (input or output).

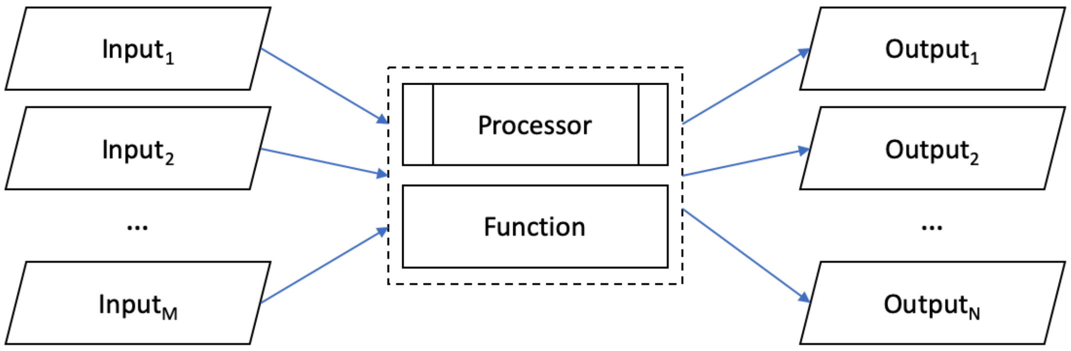

Figure 2: Basic elements for workflow execution. Source: (Li and Aydt, 2020).

A processor is a complex process which processes a data object (input data) and produces another data object (output data). It typically requires significant computational resources, memory, or storage to perform. Due to its complexity, a processor is expected to require a longer period of time to generate a response to requests. Execution of a processor is thus assumed to be asynchronous, i.e., the requesting entity will not wait for a processor to respond immediately. Instead, it will check in regular intervals if the processor has finished and if the processor's response is now available. Examples for processors include computationally heavy urban climate simulations using WRF and computational fluid dynamics simulations using ANSYS Fluent.

A function is similar to a processor in the sense that it processes input and generates output data objects just as a processor. However, it has only limited computational requirements and is expected to require only a relatively short and limited amount of time to generate a response to requests. Execution of a function is thus assumed to be synchronous, i.e., the requesting entity will wait for the function to respond. Functions are commonly found in the data pre- and post-processing stages.

A data object is a well-defined and well-structured representation of data. Data objects are used as input and generated as output by processors and functions and may come in various formats (e.g., JSON, CSV) and may be small (e.g., a few bytes) or large (e.g., several gigabytes). These are implementation specific aspects that depend on the specific use-case.

4.1.2 Workflow Composition

An arbitrary number of processes can be part of a single workflow which, similar to the aforementioned notion of a process, takes some input and produces some output. As such, a workflow can be considered as a (higher-level) process, i.e., a workflow processor. Figure 3 illustrates the concept of workflows as a composition of process (i.e., processors and functions) and data object elements.

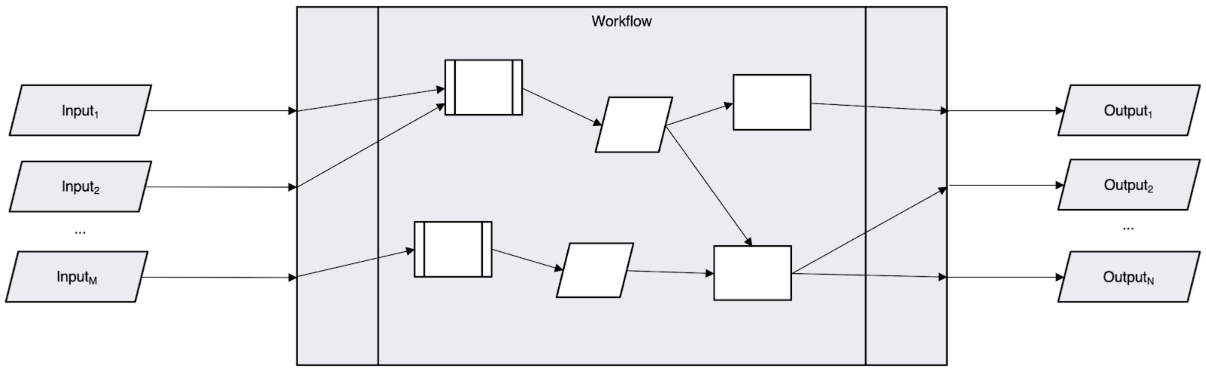

Figure 3: Workflow processor as a composition of process and data object elements.

4.2 SaaS Middleware Prototype

The general objective of the SaaS Middleware Prototype is to test the functionality needed in order to build a federation of models as required by a DUCT. An important aspect of the federation of models is the ability to carry out automated workflow execution with components deployed in a distributed manner, i.e., with components deployed on different hosts. More specifically, the prototype is used to test key services needed and the suitability of using web services as the technology of choice to realise the SaaS components.

The following principles were applied to guide the development of the SaaS Middleware Prototype:

- Specialisation instead of generalisation - The prototype should be tailored to satisfy the requirements of a DUCT Developing a general-purpose middleware standard (similar to High Level Architecture for example) is not within the scope of a DUCT.

- SaaS (Simulation-as-a-Service) instead of MSaaS (Modelling & Simulation-as-a-Service) - In the context of the DUCT, there are a number of use cases requiring a limited number of well-defined and validated workflows. For example, use cases include calculating the UHI intensity (Mughal, 2020) and evaluating the impact of electric vehicles on the urban climate (Ivanchev and Fonseca, 2020). The focus of the prototype should thus be to provide simulation services for supported workflows without the ability to allow end-users to define custom workflows and modelling. This does not mean that new use cases cannot be supported. It just means that the prototype does not support sophisticated modelling features that would be needed to allow end-users to do so on their own. It can still be done by the developers, modellers and domain experts.

Finally, it should be noted that the prototype discussed here does not yet have all the features that can be expected from a production-level software. For example, important features concerned with security (e.g., user authorisation, data encryption) are not addressed. Considering the limitations, the SaaS Middleware Prototype should not be deployed in a production environment.

4.2.1 Web Service Technology

The SaaS Middleware Prototype is using web service technology based on REST (REpresentational State Transfer), a software architectural style commonly used to provide interoperability between computer systems (Fielding, 2000). A system that conforms to the REST style (referred to as being RESTful) uses stateless protocols and standard operations. Statelessness is important as it allows the various components of the system to operate and to be understood in isolation. For example, in the context of a DUCT, a simulation request does not affect future simulation requests. They are independent. Furthermore, standard operations are important to define common interfaces and methods to interact with the corresponding services.

RESTful services make use of the HTTP protocol that powers the communication of the world wide web. The REST standard uses standard HTTP resource operations (e.g., GET or POST) to access or create resources identified by Uniform Resource Identifiers (URIs) such as `http://host:port/path/to/resource`, for example. In the context of the prototype, standard HTTP operations are used to trigger computational tasks carried out by a federate (e.g., simulation resources) that create data objects (i.e., data resources) which can then be accessed by other federates as needed. The choice of using web technology (and REST in particular) can be seen as part of a general trend in modelling and simulation to offer corresponding services in the cloud. As a result, the architecture of the prototype is similar to existing architectures, such as the one by (Wang and Wainer, 2016).

The SaaS Middleware Prototype considers two kinds of resources: services and data objects. URIs are structured accordingly whereby the service is indicated in the path component of the URI. These URIs serve as the API endpoints for interaction with a particular service. Data objects are accessed via the service that is responsible for creating them. The prototype considers two kinds of services: processor service and registry service. While there can be an arbitrary number of processor services deployed, there is only one registry service.

4.2.2 Processor Service

As explained in Section 4.1, a processor carries out complex computational tasks - such as performing simulations - using some input data and producing some output data. A processor service provides a standard API to submit, cancel and check the status of jobs as well as to access the data objects generated by this processor.

The SaaS Middleware Prototype uses descriptors (in JSON format) that define the interface of a processor in terms of input data objects needed and output data objects being generated. For example, the following example shows such a processor descriptor:

```
{
   "processor-type" : "aia",
    "input" : {
        "exposure_map": {
            "type" : "Exposure Map",
            "format" : "file"
        },
        "climate_data": {
            "type" : "Microscale Climate Data",
            "format" : "archive(tar.gz)"
        }
    },
    "output" : {
        "data_summary" : {
            "type" : "Information",
            "format" : "json"
        },
        "spatial_map": {
            "type" : "function:Information",
            "format" : "json"
        },
        "mean_variance_score": {
            "type" : "function:Information",
            "format" : "json"
        }
    }
}
```

The processor descriptor in the example above, specifies the name or type of the processor (“aia”) as well as the input (“exposure_map”, “climate_data”) and output (“data_summary”, “spatial_map”, “mean_variance_score”) data objects. In addition, for each data object, there is some information about the type and format for this data object.

A special type of processor is the workflow processor. It takes as input a workflow descriptor which defines the input and output data objects just like for any other processor. However, in addition, it also defines a set of steps that need to be performed. Each step requires execution of another processor or function. A step is considered executable if all the required input data objects exist. If that is the case, the workflow processor triggers execution of the processor or function specified by the step.

The SaaS Middleware Prototype offers the following API endpoints to interact with a processor service (the corresponding HTTP operation is indicated in brackets):

- **/processor/supported [GET]**
  Returns the type of processors that are supported by this service. Note that there can be an arbitrary number of processor services deployed on different computers (hosts). On the same computer there can be only one instance of the processor service. However, the processor service can provide support for an arbitrary number of processors. For example: `http://localhost:5000/processor/supported` will return a list of processors supported by localhost.

- **/processor/descriptor/<type> [GET]**
  Returns the descriptor (see example above) for a particular processor. For example, `http://localhost:5000/processor/descriptor/abc` would return the descriptor for processor 'abc'.

- **/processor/io_graph/<type> [GET]**
  Returns information as graphs (in JSON format) about input and output data objects (as nodes) and the processors that connect them (as edges). This is useful for inspecting workflows that may involve many data objects and processors (the prototype provides a corresponding tool for this purpose). For example, for the processor descriptor example above, the information provided by this endpoint can be visualised as illustrated in Figure 4:

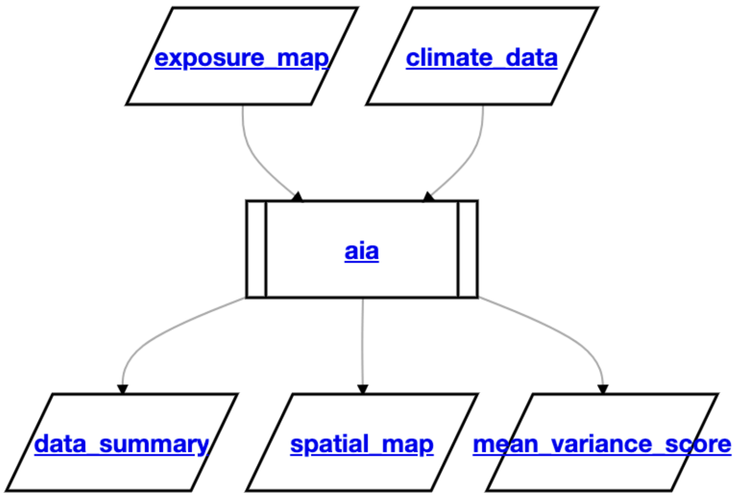

Figure 4: Graph representation of the inputs and outputs of the "aia" processor.

- **/processor/submit [POST]**
  Submits a new job to the processor. Depending on the input data objects specified in the processor descriptor, the submitting entity (either a user or - in case of workflows - a workflow controller) needs to attach the required input data objects in the required format. The expectation is that jobs may take a significant amount of time to finish. Upon successful submission, a job id is issued and returned to the caller. This job id can then be used to interact with the job as well as the data objects it produces.

- **/processor/queued [GET]**
- **/processor/running [GET]**
- **/processor/finished [GET]**
  Returns a list of jobs that are currently queued, running or finished, respectively.

- **/processor/<job_id>/cancel [DELETE]**
  Cancels a job given its job id.

- **/processor/<job_id>/status [GET]**
  Returns the status of a job given its job id.

- **/processor/<job_id>/descriptor [GET]**
  Returns the descriptor of a job given its job id. The job descriptor is largely identical with the descriptor of the processor that processes the job. In addition, it contains additional information such as the job id.

- **/processor/<job_id>/object_info/<object_id> [GET]**
  Returns information about a data object of a job given a job and object id. This is important to know if a data object is static or dynamic. In the latter case, when requesting the data object (see next endpoint), a set of parameters need to be specified. The parameters as well as their possible values are indicated in the object information returned by this endpoint.

- **/processor/<job_id>/object/<object_id> [GET or POST]**
  Returns a data object identified by its object id for a job identified by its job id. If the data object is dynamically created, parameters need to be provided by the caller of this endpoint.

Regardless of the implementation of the underlying models and/or tools, all processor components use the same interface, supporting the API endpoints described above.

4.2.3 Registry Service

The registry service keeps track of which processor services are available and where they are hosted (i.e., under what address their API endpoints can be contacted). This is particularly important for workflow execution as the workflow processor will have to make sure the various processor types needed in order to carry out the workflow are available. The SaaS Middleware Prototype offers the following API endpoints to interact with the registry service (the corresponding HTTP operation is indicated in brackets):

- **/registry/add [POST]**
  Adds an entry to the registry, i.e., registers a processor service with information about its processor descriptor and endpoint address.

- **/registry/ [DELETE]**
  Resets the entire registry, i.e., deletes all entries.

- **/registry/<type> [DELETE]**
  Deletes registry records that match the given type of processor.

- **/registry/ [GET]**
  Returns all registry records.

- **/registry/<type> [GET]**
  Returns registry records that match the given type of processor.

5 Demonstrators

In order to evaluate the SaaS Middleware Prototype, two demonstrators have been developed. The first aims at demonstrating automated workflow execution while the second aims at demonstrating the interoperability with the Cooling Singapore Decision Support System (DSS) front-end application (Chan et al., 2020).

5.1 Automated Workflow Execution

The objective of the first demonstrator is to test the ability of the SaaS Middleware Prototype to execute a workflow with a moderate level of complexity in an automated manner without the need for manual intervention. For this purpose, the UHI calculation workflow has been chosen. Firstly, it is an actual use case needed by the Cooling Singapore project which concerns itself with UHI evaluation. Secondly, the workflow to calculate the UHI requires to simulate two scenarios, an urbanised scenario and a rural reference scenario, as well as calculating the temperature difference between these two scenarios in order to obtain the UHI intensity. While only of moderate complexity, the workflow requires a variety of components, making it a good use case for testing.

UHI calculation requires execution of a mesoscale urban climate model. Cooling Singapore uses WRF for this purpose. From a technical point of view, there are two components to WRF: the WRF Pre-processing System (WPS) and the actual WRF model that execute the simulation. Both have been implemented as corresponding processor services. The WPS component performs the necessary pre-processing of the input data (in particular the local climate zone map that models the land-use). The output of the WPS component is then used by the WRF component to perform the actual simulation and generate a map with near-surface air temperature as a result. In order to obtain the UHI intensity (i.e., the temperature difference between the urban and the rural scenario), a simple function is needed that calculates a UHI intensity map based on the heat maps of the urban and rural scenario, respectively. The workflow for UHI calculation is illustrated in Figure 5.

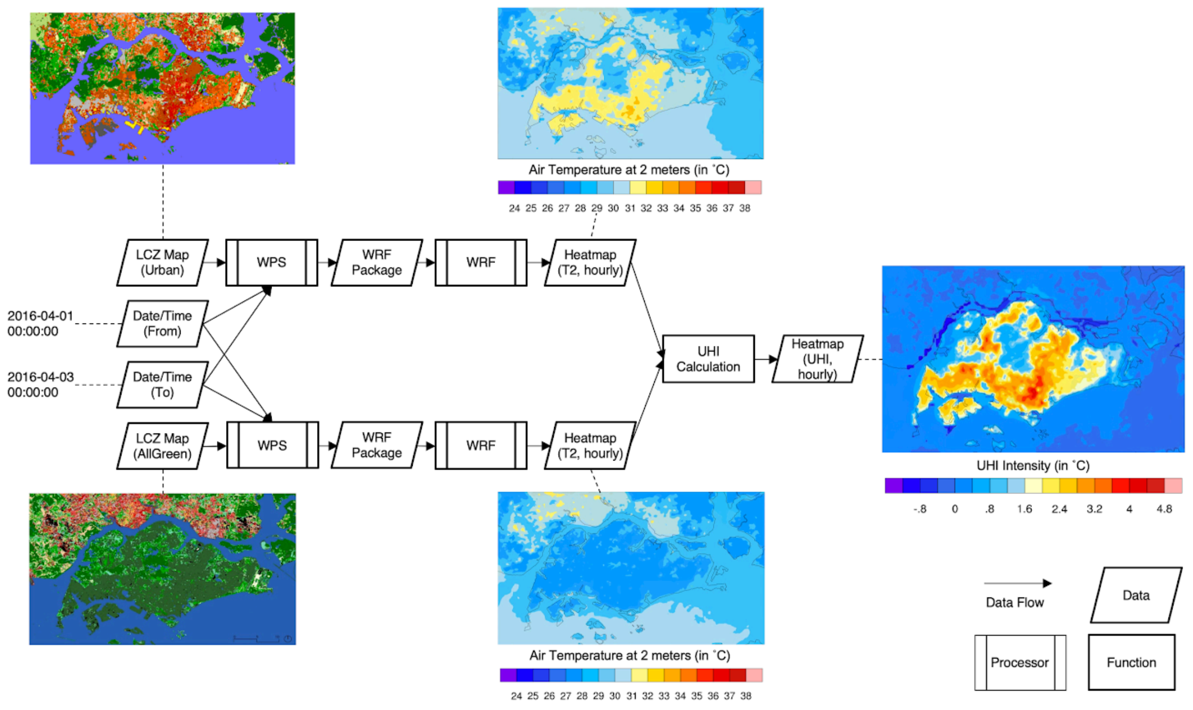

Figure 5: Overview of workflow for UHI calculation. Images: Omer Mughal, Cooling Singapore (2019).

Each of the components needed to realise the workflow has to be deployed in a suitable environment. The requirements for each component is different. Figure 6 shows the deployment configuration of the demonstrator.

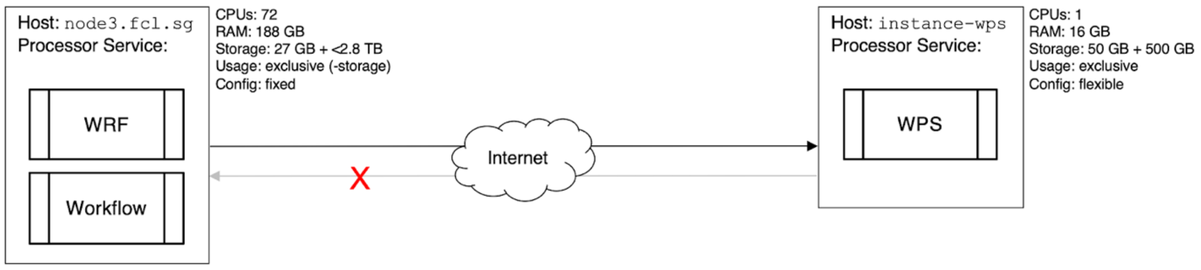

Figure 6: Deployment configuration used for the Automated Workflow Execution Demonstrator.

The pre-processing component (WPS) has high storage requirements (primarily for geographic and meteorological data) but low compute requirements as pre-processing is not using parallel computing. In addition, this component only needs to be active for pre-processing and can be shut down once the pre-processing has been completed. For the demonstrator, the WPS component has been deployed on a cloud compute instance (i.e., a virtual machine) running in the Google Cloud. The advantage is convenient availability of data storage that can be easily extended as needed and relatively low cost since the instance only needs to run for short periods of time to perform the pre-processing step for WRF.

The simulation component (WRF) has high computing and memory requirements to allow fast execution of the simulation but low storage requirements as the data produced by the simulation will only be stored temporarily on the simulation host (and moved to a dedicated data storage after the simulation has finished). This component needs to be active for an extended period of time (i.e., during the entire simulation period). For the demonstrator, the actual WRF component has been deployed on the in-house computing cluster. The advantage is a larger number of computing resources (i.e., CPUs and memory) for an extended period of time (e.g., days) at fixed operating costs.

The workflow processor component (Workflow) only coordinates the process by triggering the various services and moves data objects as needed. This component does not require a significant amount of compute, memory or storage resources. However, it needs to be able to communicate with the other components. While the in-house computing cluster can access the internet, it is not possible the other way around. The workflow processor has thus been deployed on the same host as the WRF component. For a production environment, a virtual private network configuration would likely be the best option. For the demonstrator this was deemed unnecessary as the workflow processor is able to connect to the WPS component using the internet without the need for a virtual private network.

This demonstrator shows the capabilities of the SaaS Middleware Prototype with respect to performing automated workflow executions. Because a workflow processor does not need to know anything about the way a processor works internally, it should also be possible to create nested workflows. This refers to the case whereby a step that is part of an ‘outer’ workflow may trigger another workflow execution. In principle, this allows for great flexibility with respect to the composability of processor components. However, nested workflow execution has not been tested and is subject to future work.

5.2 DSS Interoperability

The objective of the second demonstrator is to test the ability of the SaaS Middleware Prototype to interact with the Cooling Singapore DSS front-end application and provide the necessary flexibility to support a representative DSS use-case. For the purpose of testing interoperability, a use-case has been chosen that requires the DSS to compare different scenarios concerned with urban design measures aimed at improving outdoor thermal comfort. This use-case involves executing a number of microscale urban climate simulations to produce climate data. This data is then further processed to generate useful statistics for the end user. For testing interoperability between the prototype and the DSS, the simulation part is not needed and has therefore not been included in the demonstrator. Instead, the demonstrator focuses on the SaaS component (AIA - see below) that generates the statistics and interacts with the DSS front-end application.

Each urban design scenario has been simulated using microscale urban climate models (not part of this demonstrator). The climate data output of these scenarios is then processed by an Action Impact Analyser (AIA) component. This component calculates some statistics and aggregates the information in form of a spatial map and some mean-variance scores. The scores are then used by the DSS application for visualisation purposes as illustrated in Figure 7.

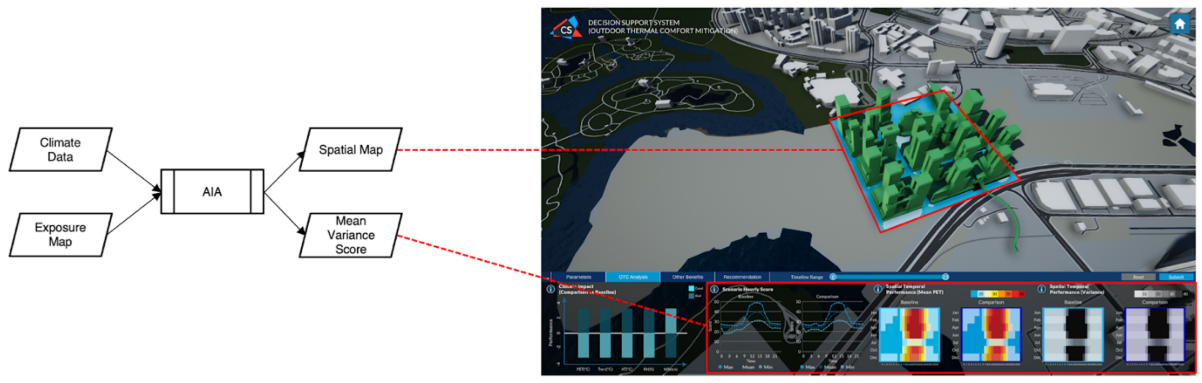

Figure 7: Overview of the AIA component and how the output data objects (spatial map and mean variance score) are used by the DSS for visualisation purposes. Image: Ye Cong, Cooling Singapore, 2020.

The deployment of this demonstrator is simple as there is only one component. Figure 8 shows the deployment configuration of the DSS Interoperability demonstrator.

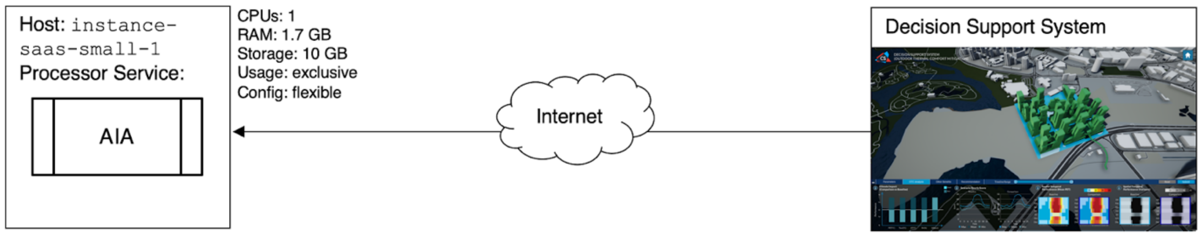

Figure 8: Deployment configuration used for the DSS Interoperability Demonstrator.

The AIA component does not have significant compute, memory or storage requirements. For simplicity, the processor has been deployed on a small, inexpensive Google Cloud instance. The DSS can access the AIA component via the internet.

Unlike the previous demonstrator which produced static data objects (i.e., heatmaps), this demonstrator produces dynamic data objects. To be more specific, when requesting any of the two data objects (spatial map or mean-variance score), the DSS provides a set of parameters, depending on input provided by the DSS user. Depending on these parameters, the data objects are generated dynamically upon request. A particular challenge in this regard has been the time needed to re-calculate the data objects. Performance issues were addressed by performing efficient calculations and data caching mechanisms. However, the general challenge of having to re-calculate data objects remains. The issue here is that without knowing in advance what parameters may be of interest to users, it is not possible to perform these calculations in advance. Optimistically calculating data objects with many possible parameters values is also not a good idea as it would be very wasteful in terms of storage requirements and in many cases, simply not possible due to the large number of possible parameter combinations.

6 Conclusions and Future Work

The SaaS Middleware Prototype has been developed to test how web service technology (and REST technology in particular) can be used to build a middleware for a federation of loosely coupled models specifically for the purpose of building a DUCT. Two key aspects that were tested involve the automated execution of workflows as well as interoperability with an end-user application involving user-driven parameterised data object requests. Testing with the prototype confirmed that the use of web service technology (in particular REST) is a suitable choice of technology in order to realise a DUCT.

Future work will focus on the development of a DUCT using a SaaS middleware. For this purpose, the current prototype will be developed further and enhanced to address open issues. For example, this includes adding a data repository service that allows for convenient storage of data objects in a centralised location. Currently data objects are stored temporarily with the processors that created them and eventually deleted. Furthermore, future work will also provide support for handling stochastic models and aggregation of output data generated by the various simulation runs. In order for the SaaS Middleware to be suitable for a production environment, security features (e.g., user management and data encryption) will have to be developed.

7 References

Chan, M., Ye, C., Perhac, J. (2020). Decision Support System: User research, usability analysis and computational build. DOI: https://doi.org/10.3929/ethz-b-000450016.

Fielding, R.T. (2000). Chapter 5: Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

IEEE Standard 1516-2010. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. DOI: https://doi.org/10.1109/IEEESTD.2010.5553440.

Ivanchev, J., Fonseca, J.A. (2020). Anthropogenic Heat Due to Road Transport: A Mesoscopic Assessment and Mitigation Potential of Electric Vehicles and Autonomous Vehicles in Singapore. DOI: https://doi.org/10.3929/ethz-b-000401288.

Kayanan, D.R., Fonseca, J.A., Norford, L.K. (2020). Anthropogenic Heat of Power Generation in Singapore: analyzing today and a future electro-mobility scenario. DOI: https://doi.org/10.3929/ethz-b-000412794.

Kayanan, D.R., Resende Santos, L.G., Ivanchev, J., Fonseca J.A., and Norford, L.K. (2019). Anthropogenic Heat Sources in Singapore. DOI: https://doi.org/10.3929/ethz-b-000363683.

Li, S., Aydt, H. (2020). System Analysis of Mesoscale and Microscale Urban Climate Simulation Workflows. DOI: https://doi.org/10.3929/ethz-b-000432258.

Mughal, M.O. (2020). Modelling the Urban Heat Island in Singapore – state of the art WRF model technical details. DOI: https://doi.org/10.3929/ethz-b-000412358.

Wang, S.X. and Wainer, G. (2016). Modeling and simulation as a service architecture for deploying resources in the Cloud. International Journal of Modeling, Simulation, and Scientific Computing, Vol 7(1), 2016. DOI: https://doi.org/10.1142/S1793962316410026.